Spoofing certificates with MD5 collisions - party like it's 2008!

Tomer Peled
Yoni Rozenshein

DEF CON 31
Crypto & Privacy Village

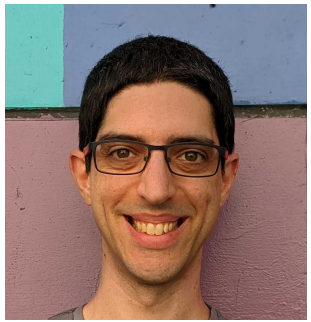
Spoofing certificates with MD5 collisions – party like it's 2008!

Tomer Peled
Yoni Rozenshein

$ whoami /all

Tomer Peled

- Security & vulnerability researcher at Akamai
- Gamer
- Krav Maga practitioner
- github.com/tomerpeled92
- @TomerPeled92



Yoni Rozenshein

- Software engineer at Akamai
- OS internals, math & cryptography enthusiast
- @1yoni

CVE-2022-34689

Windows CryptoAPI Spoofing Vulnerability
CVE-2022-34689

Released: Oct 11, 2022

Assigning CNA: Microsoft

CVE-2022-34689

Exploitability

The following table provides an exploitability assessment for this vulnerability at the time of original publication.

Publicly Disclosed	Exploited	Latest Software Release
No	No	Exploitation More Likely

FAQ

What is the nature of the spoofing?

An attacker could manipulate an existing public x.509 certificate to spoof their identity and perform actions such as authentication or code signing as the targeted certificate.

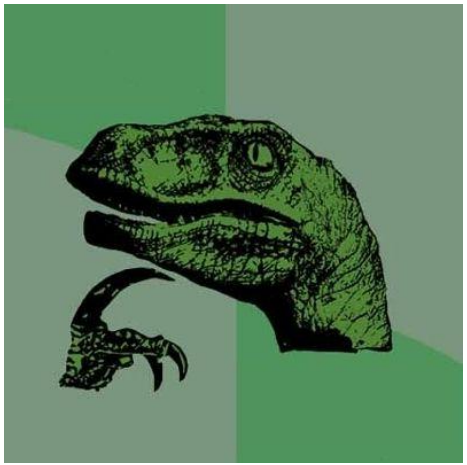
Acknowledgements

UK National Cyber Security Centre (NCSC) and the National Security Agency (NSA)

Microsoft recognizes the efforts of those in the security community who help us protect customers through coordinated vulnerability disclosure. See Acknowledgements for more information.

What is CryptoAPI?

- Microsoft’s interface for handling certificates in Windows
- Responsible for encryption and decryption
- **Validation of certificates**
- It is mainly implemented in crypt32.dll



Certificate chain & validation

Root CA (e.g. DigiCert Global Root CA)

Signed by

Intermediate CA (e.g. DigiCert TLS RSA SHA256 2020 CA1)

Signed by

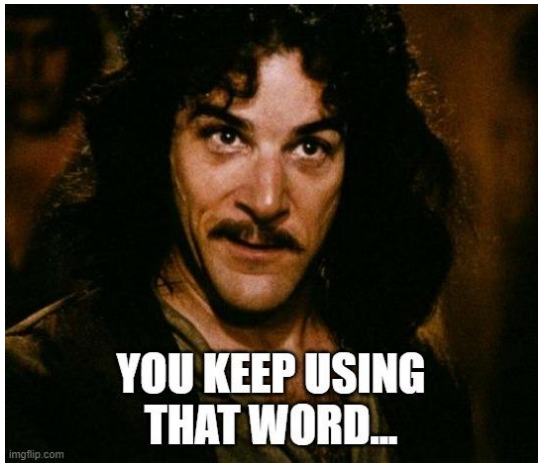
End certificate (e.g. www.akamai.com)
This is the certificate that is being validated

Why is certificate validation important?

Who uses CryptoAPI?

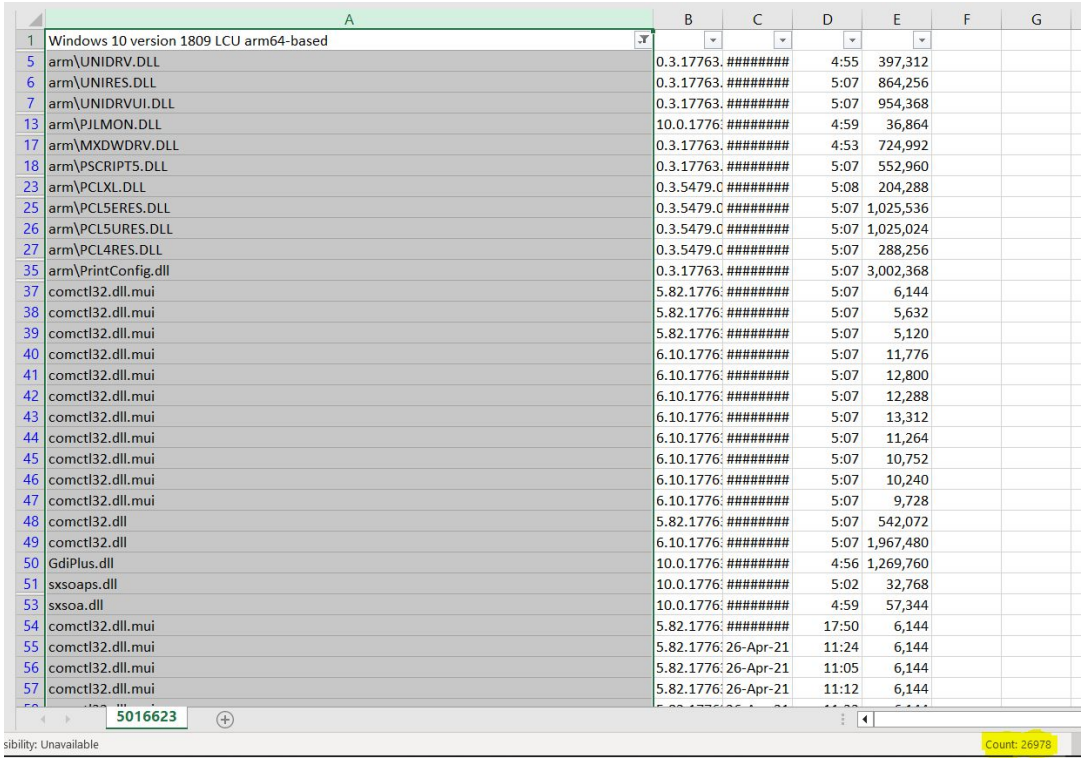
- Windows - code signing
- Some browsers - such as Chrome and Edge
- Data transfer clients - curl, wget
- FTP managers
- EDRs!

Many other applications.



Locating the vulnerability

Locating the fix (≠ vulnerability)



	A	B	C	D	E	F	G
1	Windows 10 version 1809 LCU arm64-based						
5	arm\UNIDRV.DLL	0.3.17763.	########	4:55	397,312		
6	arm\UNIRES.DLL	0.3.17763.	########	5:07	864,256		
7	arm\UNIDRVUI.DLL	0.3.17763.	########	5:07	954,368		
13	arm\PJLMON.DLL	10.0.1776:	########	4:59	36,864		
17	arm\MXDWDRV.DLL	0.3.17763.	########	4:53	724,992		
18	arm\PSCRIPT5.DLL	0.3.17763.	########	5:07	552,960		
23	arm\PCLXL.DLL	0.3.5479.0	########	5:08	204,288		
25	arm\PCL5ERES.DLL	0.3.5479.0	########	5:07	1,025,536		
26	arm\PCL5URES.DLL	0.3.5479.0	########	5:07	1,025,024		
27	arm\PCL4RES.DLL	0.3.5479.0	########	5:07	288,256		
35	arm\PrintConfig.dll	0.3.17763.	########	5:07	3,002,368		
37	comctl32.dll.mui	5.82.1776:	########	5:07	6,144		
38	comctl32.dll.mui	5.82.1776:	########	5:07	5,632		
39	comctl32.dll.mui	5.82.1776:	########	5:07	5,120		
40	comctl32.dll.mui	6.10.1776:	########	5:07	11,776		
41	comctl32.dll.mui	6.10.1776:	########	5:07	12,800		
42	comctl32.dll.mui	6.10.1776:	########	5:07	12,288		
43	comctl32.dll.mui	6.10.1776:	########	5:07	13,312		
44	comctl32.dll.mui	6.10.1776:	########	5:07	11,264		
45	comctl32.dll.mui	6.10.1776:	########	5:07	10,752		
46	comctl32.dll.mui	6.10.1776:	########	5:07	10,240		
47	comctl32.dll.mui	6.10.1776:	########	5:07	9,728		
48	comctl32.dll	5.82.1776:	########	5:07	542,072		
49	comctl32.dll	6.10.1776:	########	5:07	1,967,480		
50	GdiPlus.dll	10.0.1776:	########	4:56	1,269,760		
51	sxsoaps.dll	10.0.1776:	########	5:02	32,768		
53	sxsoa.dll	10.0.1776:	########	4:59	57,344		
54	comctl32.dll.mui	5.82.1776:	########	17:50	6,144		
55	comctl32.dll.mui	5.82.1776:	26-Apr-21	11:24	6,144		
56	comctl32.dll.mui	5.82.1776:	26-Apr-21	11:05	6,144		
57	comctl32.dll.mui	5.82.1776:	26-Apr-21	11:12	6,144		

5016623

sibility: Unavailable

Count: 26978

Locating the fix in *crypt32.dll* with BinDiff

1.00	0.99	000000018015...	__imp_NCryptFinalizeKey	Imported	0000000180152708	__imp_NCryptFin...	Imported	-1	0	-1	-1	0	-1				
1.00	0.99	000000018015...	__imp_NCryptDeleteKey	Imported	0000000180152710	__imp_NCryptDel...	Imported	-1	0	-1	-1	0	-1				
1.00	0.99	000000018015...	__imp_NCryptSecretAgreement	Imported	0000000180152718	__imp_NCryptSec...	Imported	-1	0	-1	-1	0	-1				
1.00	0.99	000000018015...	__imp_NCryptProtectSecret	Imported	0000000180152720	__imp_NCryptPro...	Imported	-1	0	-1	-1	0	-1				
1.00	0.99	000000018015...	__imp_NCryptDecrypt	Imported	0000000180152728	__imp_NCryptDec...	Imported	-1	0	-1	-1	0	-1				
1.00	0.99	000000018015...	__imp_GetBasicProfileFolderPath	Imported	0000000180152738	__imp_GetBasicP...	Imported	-1	0	-1	-1	0	-1				
1.00	0.99	000000018015...	__imp_GetAppContainerRegistryHandle	Imported	0000000180152740	__imp_GetAppCon...	Imported	-1	0	-1	-1	0	-1				
0.94	0.99	000000018004...	CCertChainEngine::CreateChainCont...	Normal	0000000180041A68	CCertChainEngin...	Normal	0	140	9	18	197	33				

Diffing the patched function with IDA

```
IssuerObject = CCertObjectCache::FindIssuerObject(v15, *v16, &v106);
  v81 = IssuerObject;
  v9 = IssuerObject;
  if ( IssuerObject )
  {
    v18 = *(IssuerObject + 11);
    v19 = *(v18 + 16);
  }
  v20 = 1;
  if ( !v9 )
  {
    EndObjectByHash = CCertObjectCache::FindEndObjectByHash(*(v5 + 23),
pvData);
    v81 = EndObjectByHash;
    v9 = EndObjectByHash;
    if ( !EndObjectByHash )
      goto LABEL_23;
    v22 = *(EndObjectByHash + 11);
    v23 = *(v22 + 16);
```

```
IssuerObject = CCertObjectCache::FindIssuerObject(v15, *v16, &v106);
  v81 = IssuerObject;
  v9 = IssuerObject;
  if ( IssuerObject )
  {
    v18 = *(IssuerObject + 11);
    v19 = *(v18 + 16);
    if ( v19 != v6->cbCertEncoded || memcmp_0(*(v18 + 8), v6-
>pbCertEncoded, v19) )
    {
      CCertObject::Release(v9);
      v9 = 0i64;
      v81 = 0i64;
    }
  }
  v20 = 1;
  if ( !v9 )
  {
    EndObjectByHash = CCertObjectCache::FindEndObjectByHash(*(v5 + 23),
pvData);
    v81 = EndObjectByHash;
    v9 = EndObjectByHash;
    if ( !EndObjectByHash )
      goto LABEL_23;
    v22 = *(EndObjectByHash + 11);
    v23 = *(v22 + 16);
    if ( v23 != a3->cbCertEncoded || memcmp_0(*(v22 + 8), a3-
>pbCertEncoded, v23) )
    {
      CCertObject::Release(v9);
      v9 = 0i64;
      v81 = 0i64;
    }
```

memcmp who?!

This is a **cache lookup** routine!

If a certificate object is found in cache…

- **Before the patch**: it's returned
- **After the patch**: it's *memcmp*'d to the input object, on success – returned

Reaching the vulnerable code

Reaching the vulnerable code

```
__int64 __fastcall CCertChainEngine::CreateChainContextFromPathGraph(
        CCertChainEngine *this,
        struct CChainCallContext *a2,
        const struct _CERT_CONTEXT *a3,
        void *a4,
        const struct _CERT_CHAIN_CONTEXT **a5)
```

Xref

```
__int64 __fastcall CCertChainEngine::GetChainContext(
        CCertChainEngine *this,
        const struct _CERT_CONTEXT *a2,
        struct _FILETIME *a3,
        void *a4,
        struct _CERT_CHAIN_PARA *a5,
        unsigned int a6,
        void *a7,
        const struct _CERT_CHAIN_CONTEXT **a8)
```

Xref

```
BOOL __stdcall CertGetCertificateChain(
        HCERTCHAINENGINE hChainEngine,
        PCCERT_CONTEXT pCertContext,
        LPFILETIME pTime,
        HCERTSTORE hAdditionalStore,
        PCERT_CHAIN_PARA pChainPara,
        DWORD dwFlags,
        LPVOID pvReserved,
        PCCERT_CHAIN_CONTEXT *ppChainContext)
```

CertGetCertficateChain prototype

C++ Copy

```
BOOL CertGetCertificateChain(
  [in, optional] HCERTCHAINENGINE     hChainEngine,
  [in]           PCCERT_CONTEXT       pCertContext,
  [in, optional] LPFILETIME           pTime,
  [in]           HCERTSTORE           hAdditionalStore,
  [in]           PCERT_CHAIN_PARA     pChainPara,
  [in]           DWORD                dwFlags,
  [in]           LPVOID               pvReserved,
  [out]          PCCERT_CHAIN_CONTEXT *ppChainContext
);
```

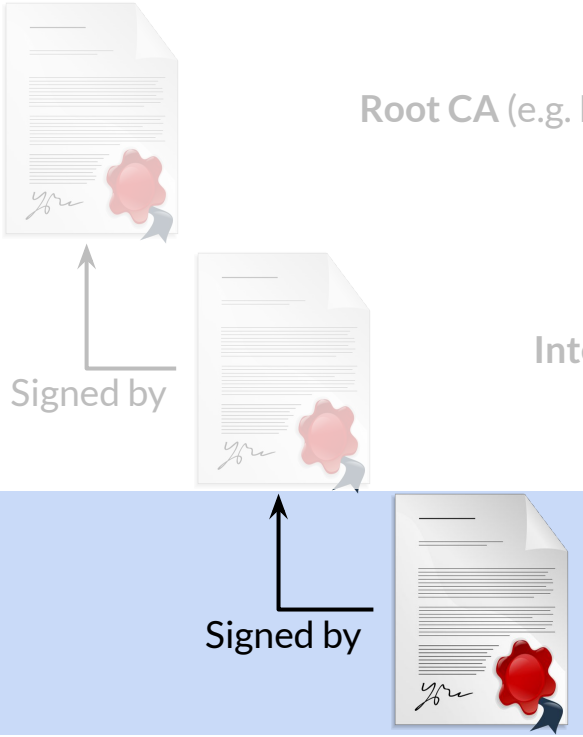
dwFlags: CERT_CHAIN_CACHE_END_CERTIFICATE

C++ Copy

```
BOOL CertGetCertificateChain(
  [in, optional] HCERTCHAINENGINE     hChainEngine,
  [in]           PCCERT_CONTEXT       pCertContext,
  [in, optional] LPFILETIME           pTime,
  [in]           HCERTSTORE           hAdditionalStore,
  [in]           PCERT_CHAIN_PARA     pChainPara,
  [in]           DWORD                dwFlags,
  [in]           LPVOID               pvReserved,
  [out]          PCCERT_CHAIN_CONTEXT *ppChainContext
);
```

Value	Meaning
CERT_CHAIN_CACHE_END_CERT 0x00000001	When this flag is set, the end certificate is cached, which might speed up the chain-building process. By default, the end certificate is not cached, and it would need to be verified each time a chain is built for it.

Certificate chain & validation - reminder

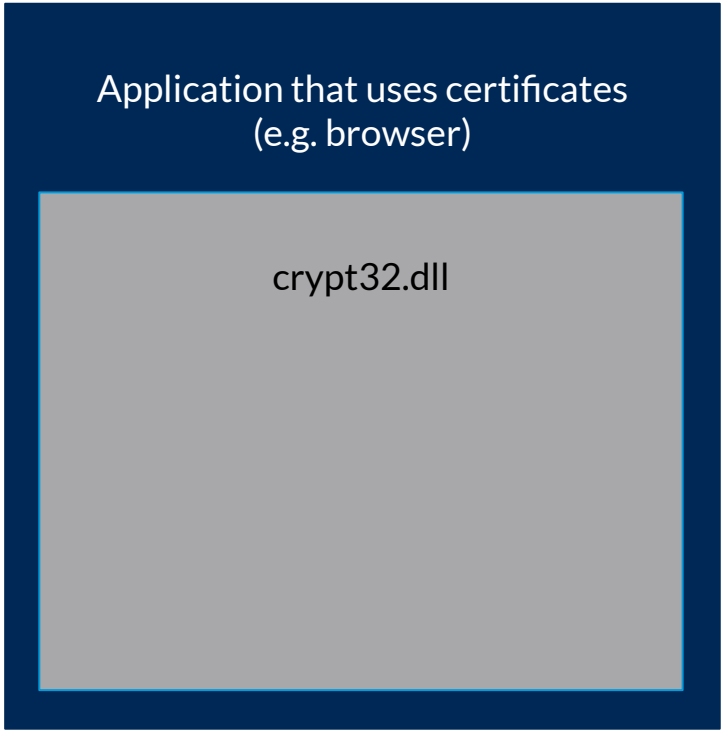


End-certificate cache

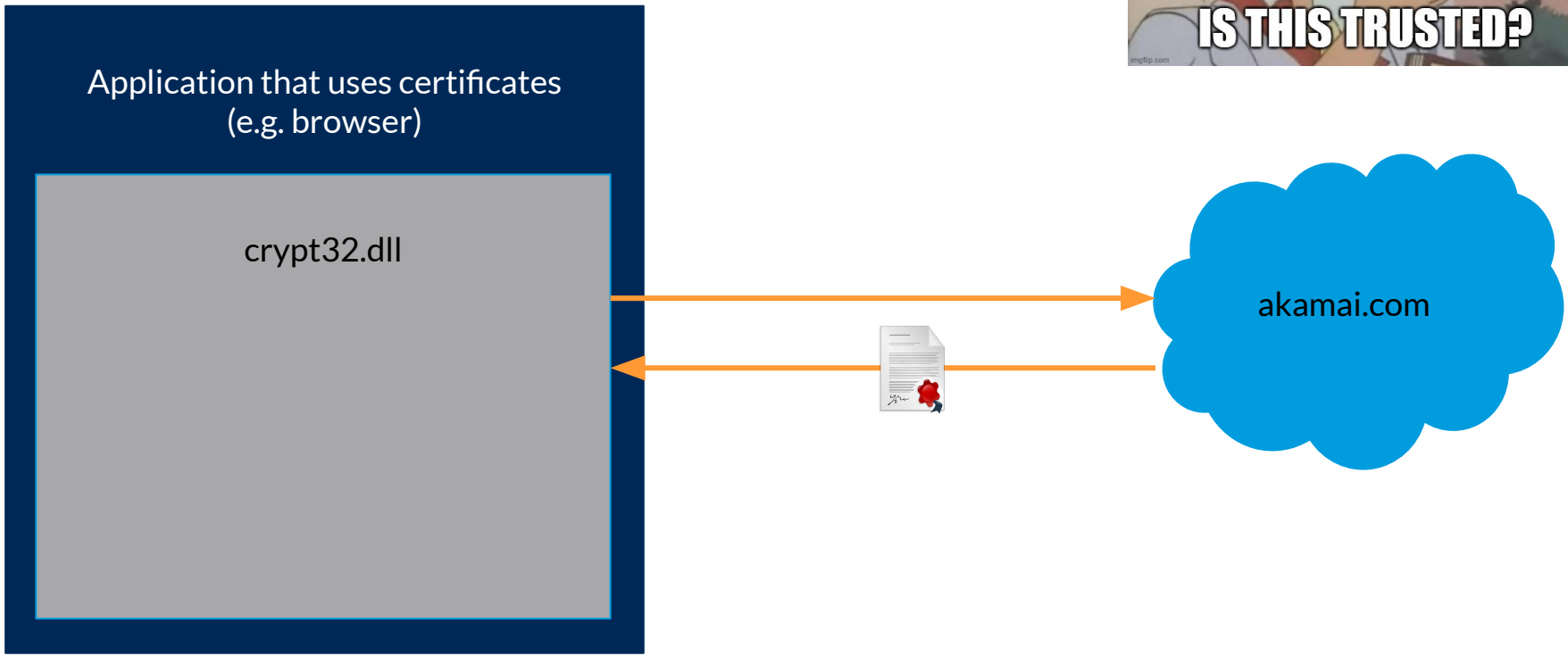
- Hashmap containing end-certificate objects
- Uses their MD5 as an indexer [💡 !!!]

```
Entry = hashmap_start + 16*(CertObjectCacheHashMd5Identifier(Cert) % 128)
```

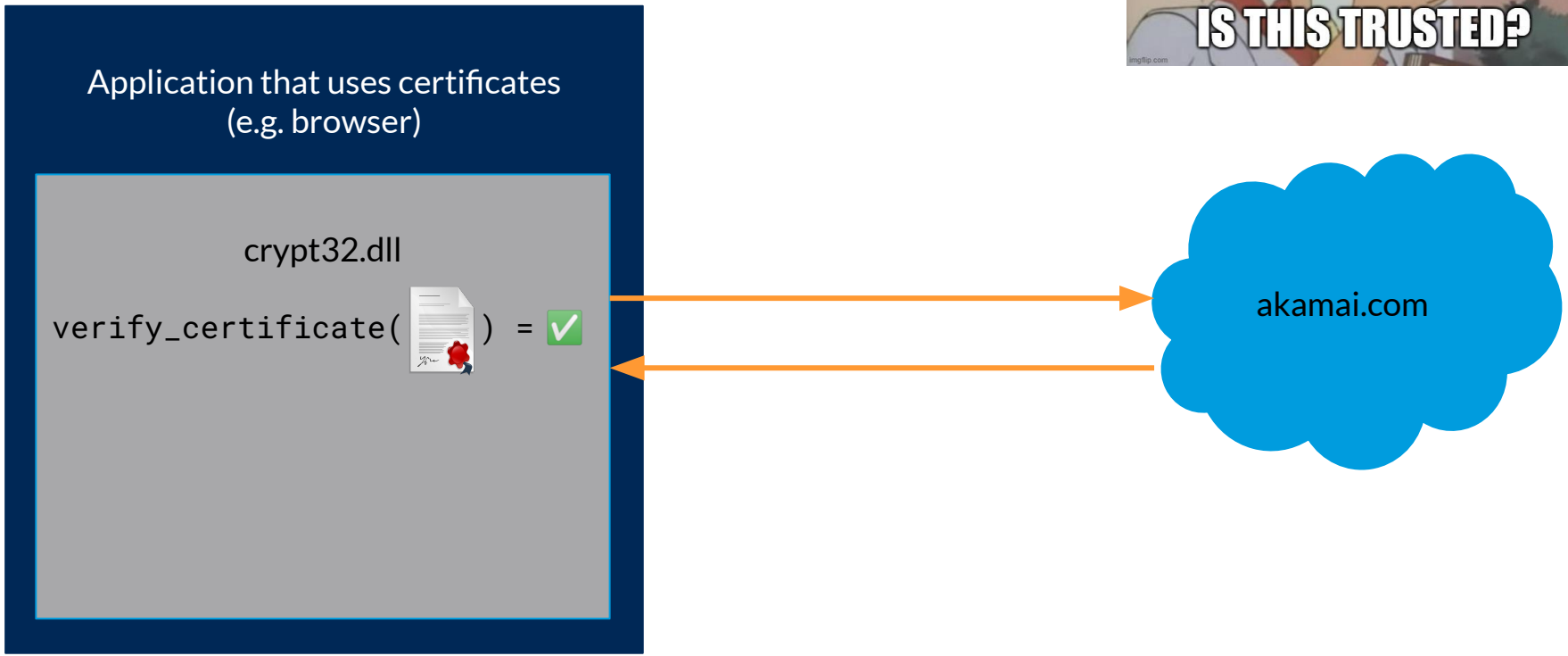
How the certificate validation works



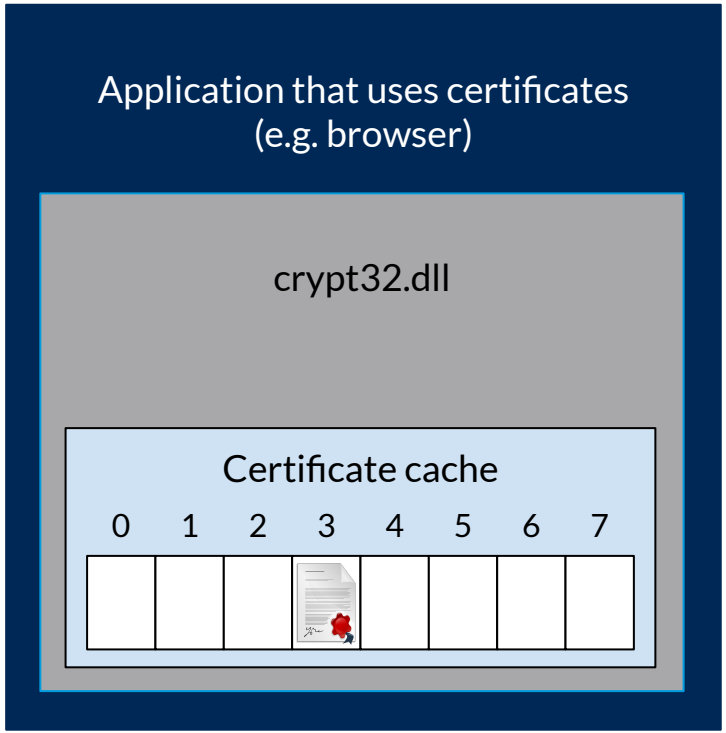
How the certificate validation works



How the certificate validation works



How the certificate validation works



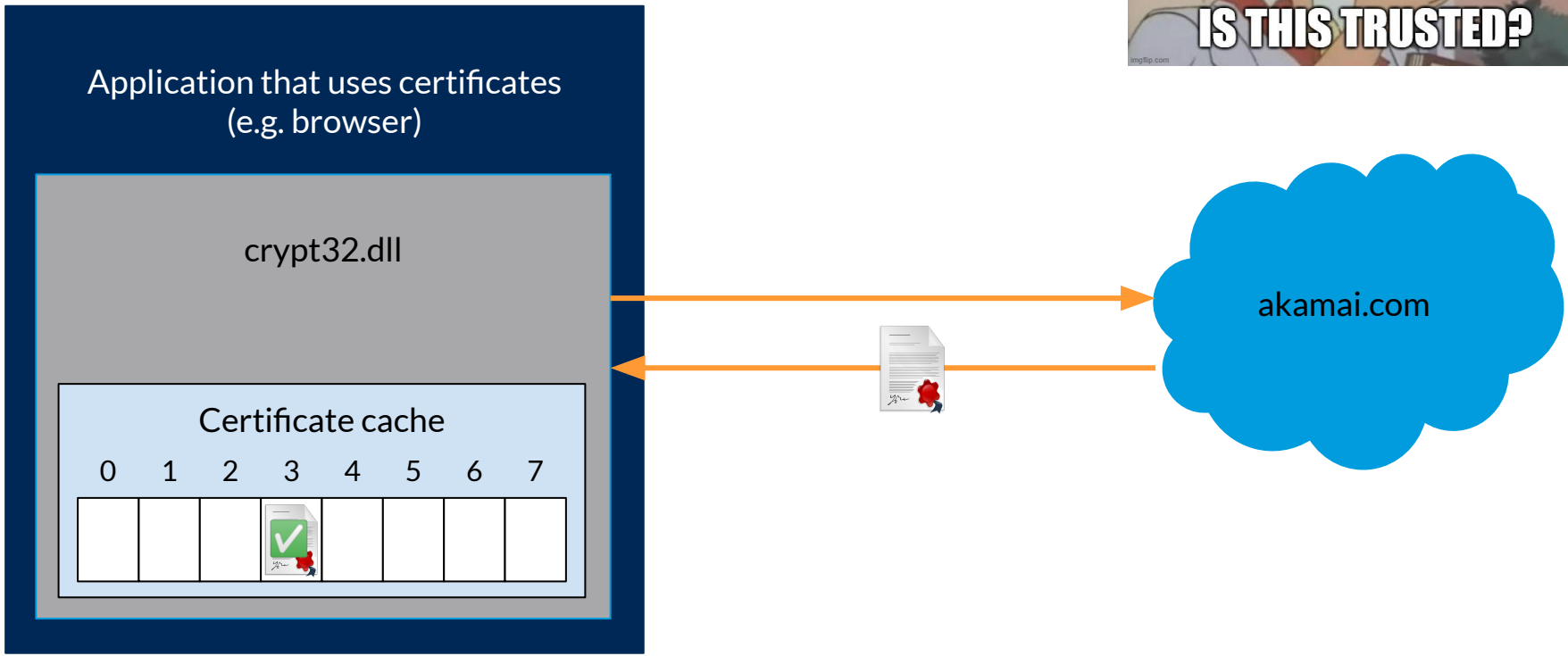
`MD5(`  `) =`

`fc2baa1a20b4d5190b122b383d744983`

`83 → (mod 128) → 3`

Next time *crypt32* sees a certificate with the same MD5, it will be immediately verified and trusted

How the certificate validation works



Validation process recap

$Cert_{cached}$ = certificate_cache[**MD5**($Cert_{input}$)]

Unpatched version: $Cert_{cached}$ **exists**

Patched version: memcmp($Cert_{input}$, $Cert_{cached}$)

Running CertGetCertificateChain with 2 certificates

```
loop number: 1
Enter certificate path: C:\TEMP\final\msft_root.cer
Cert file opened!
Cert file read to buffer!
A new certificate Context has been created.
cert size of the cert is: 0000000000000DF5
cert size of the cert in context: 0000000000000DF5
The chain has been created.
The size of the chain context is 72.
1 simple chains found.

Error status for the chain:0
No error found for this certificate or chain.
```

```
loop number: 2
Enter certificate path: C:\TEMP\final\evil_root.cer
Cert file opened!
Cert file read to buffer!
A new certificate Context has been created.
cert size of the cert is: 0000000000000DF5
cert size of the cert in context: 0000000000000DF5
The chain has been created.
The size of the chain context is 72.
1 simple chains found.

Error status for the chain:0
No error found for this certificate or chain.
```

```
C:\TEMP\final>certutil -hashfile msft_coll.cer MD5
MD5 hash of msft_coll.cer:
bb04b2abc1972393c2778f96621d8ea2
CertUtil: -hashfile command completed successfully.
```

```
C:\TEMP\final>certutil -hashfile evil_coll.cer MD5
MD5 hash of evil_coll.cer:
bb04b2abc1972393c2778f96621d8ea2
CertUtil: -hashfile command completed successfully.
```

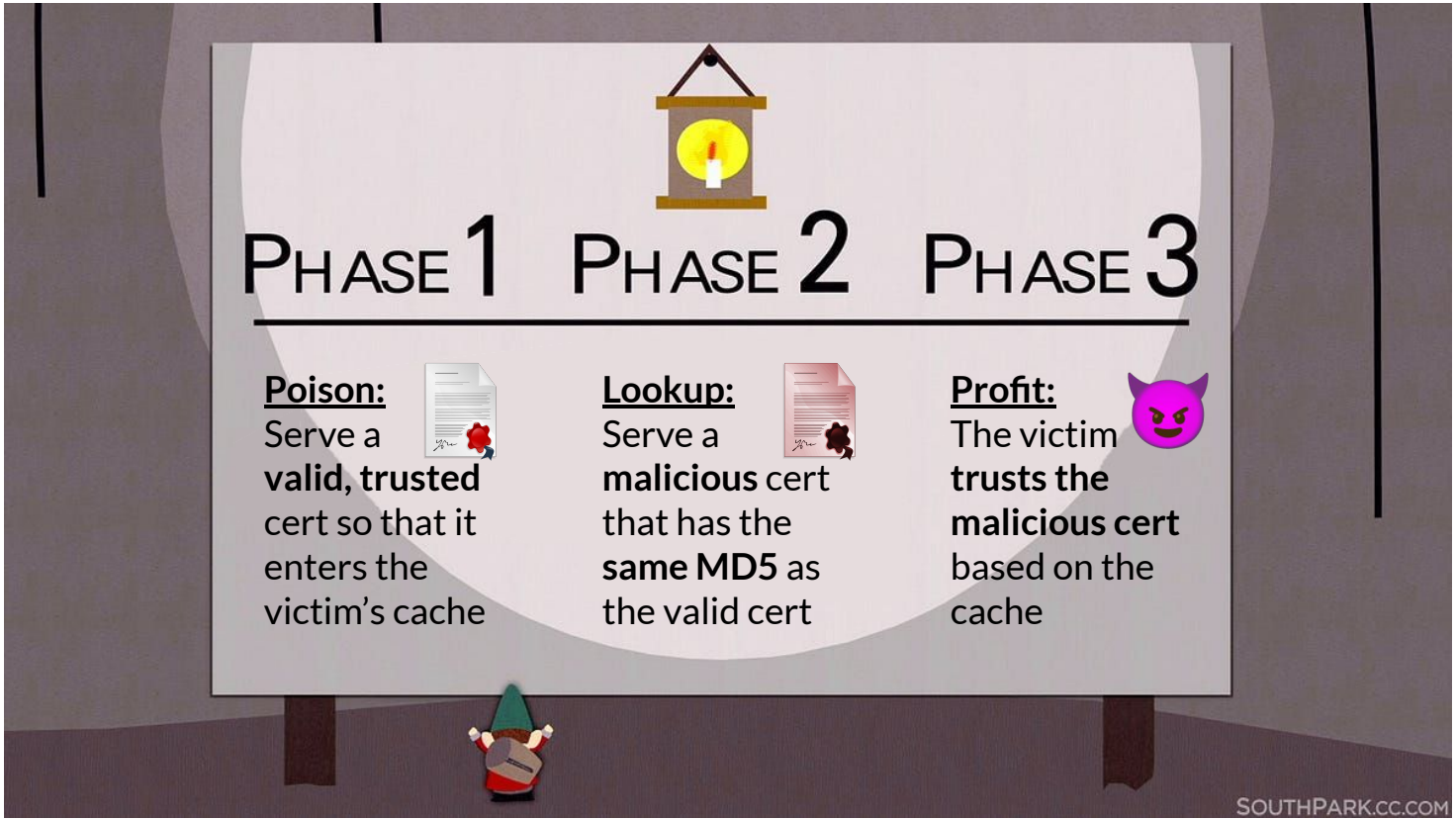
High-level attack flow

The attacker serves a malicious certificate with the same MD5 thumbprint as a valid one in CryptoAPI's cache

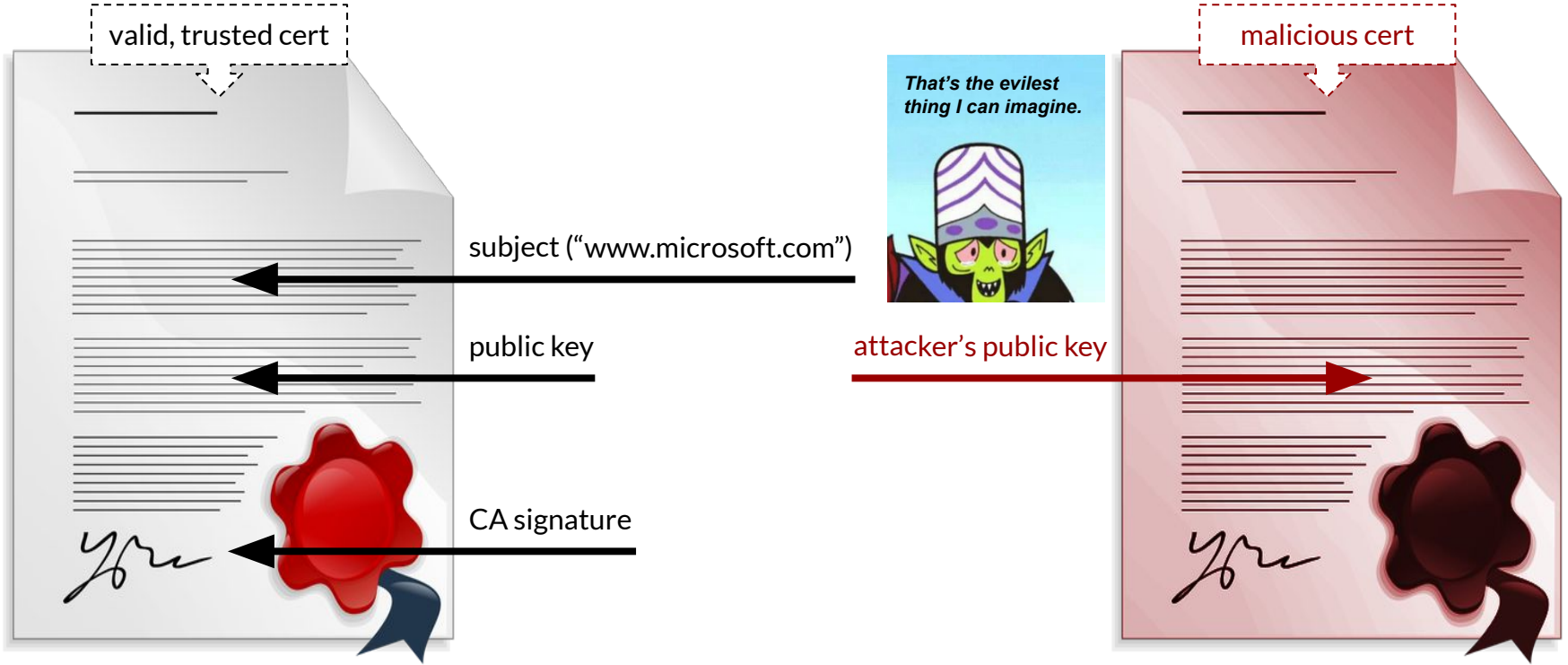
The vulnerable application compares the identical MD5 thumbprints

The victim trusts the attacker

Theoretical attack phases



What's in the malicious cert?



Can we actually build a **malicious** cert?

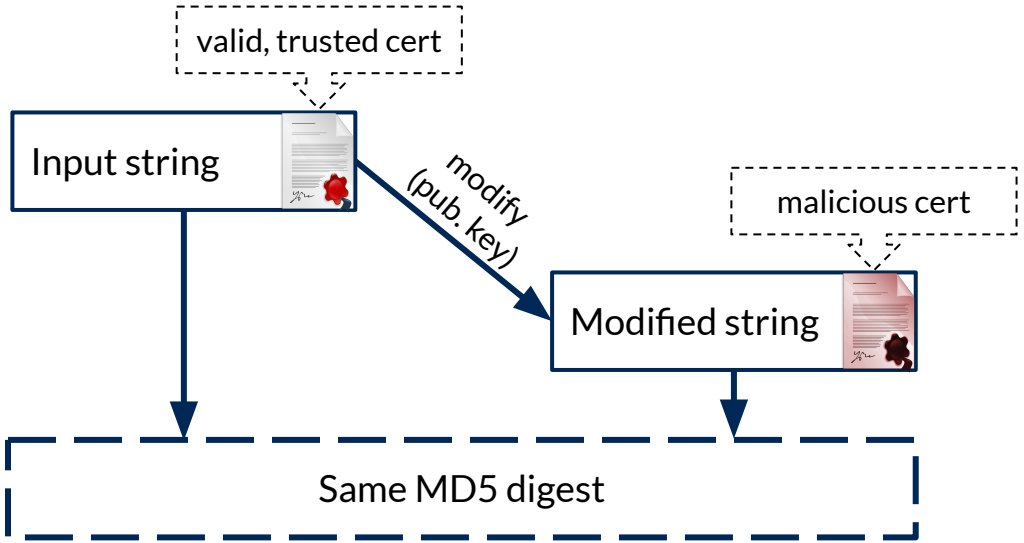
MD5() = MD5()

Second-preimage attack on MD5?

1. Choose:

2. Compute:

3. Conquer:



We don’t know how to *feasibly* compute an MD5 **second-preimage**.

We only know how to compute MD5 **collisions**.

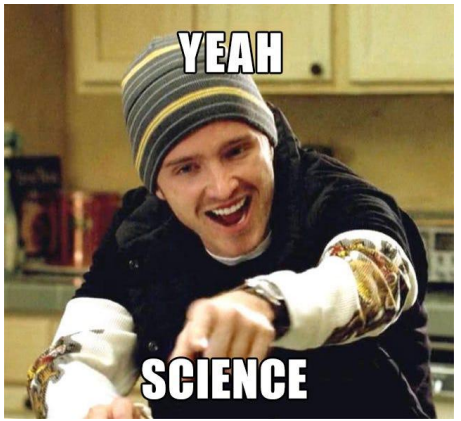
MD5 considered harmful 15 years ago

HashClash

Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate

Marc Stevens[1], Alexander Sotirov[2], Jacob Appelbaum[3], Arjen Lenstra[4,5], David Molnar[6], Dag Arne Osvik[4] and Benne de Weger[7]

[1] CWI, Amsterdam, The Netherlands
[2] http://www.phreedom.org
[3] http://www.appelbaum.net
[4] EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
[5] Alcatel-Lucent Bell Laboratories
[6] University of California at Berkeley
[7] EiPSI, TU Eindhoven, The Netherlands
[1–7] md5-collisions@phreedom.org



Collision attacks on MD5

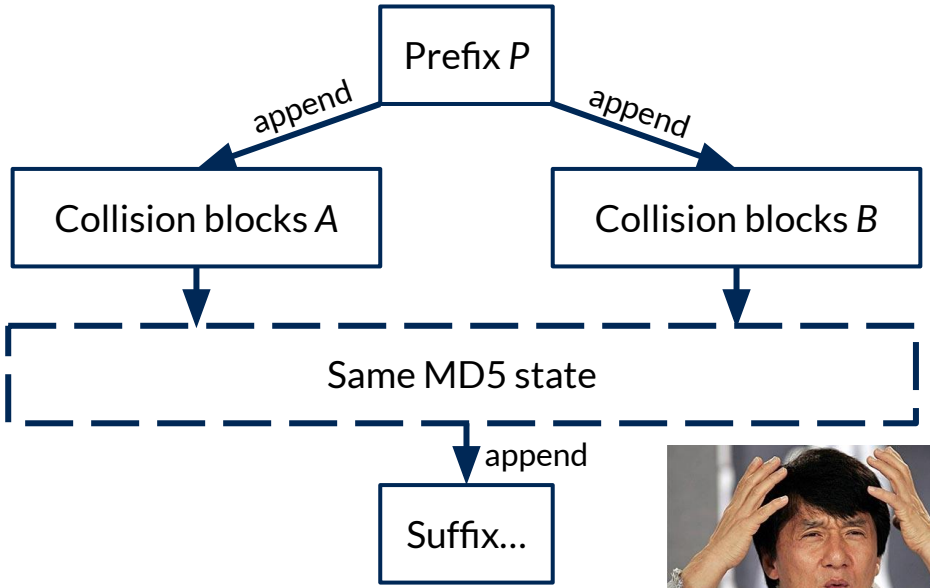
Simple (identical-prefix) collision: $\mathrm{MD5}(P \,||\, A) = \mathrm{MD5}(P \,||\, B)$

In 2023… a few seconds ⚡ on a laptop.

1. Choose:

2. Compute:

3. Conquer:



Demo #1 - simple collision using “fastcoll”

```
# Specify a prefix
$ echo "Once upon a time," > prefix

# Compute the collision
$ docker run --rm -it -v $PWD:/work -w /work -u $UID:$GID brimstone/fastcoll
--prefixfile prefix -o msg1.bin msg2.bin

# Different files, same MD5. Notice the "collision blocks"
$ xxd msg1.bin
$ xxd msg2.bin
$ md5sum msg1.bin msg2.bin

# We can add a suffix after the collision blocks
$ echo "And they lived happily ever after." | tee -a msg1.bin msg2.bin
$ xxd msg1.bin
$ xxd msg2.bin
$ md5sum msg1.bin msg2.bin
```

Collision attacks on MD5

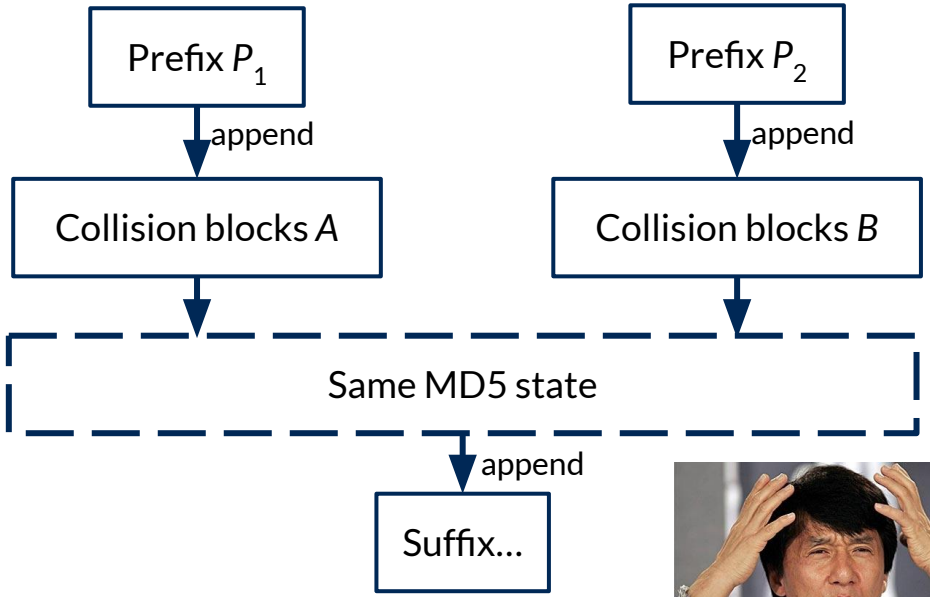
Chosen-prefix collision: $\mathrm{MD5}(P_1 \,||\, A) = \mathrm{MD5}(P_2 \,||\, B)$

In 2023… a few hours ⏳ on a single cloud instance with a GPU.

1. Choose:

2. Compute:

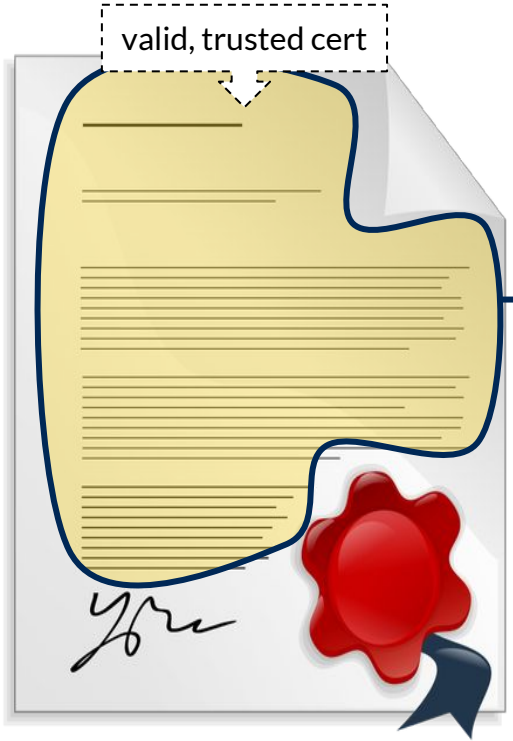
3. Conquer:



Can we use MD5 collisions to construct a malicious cert?

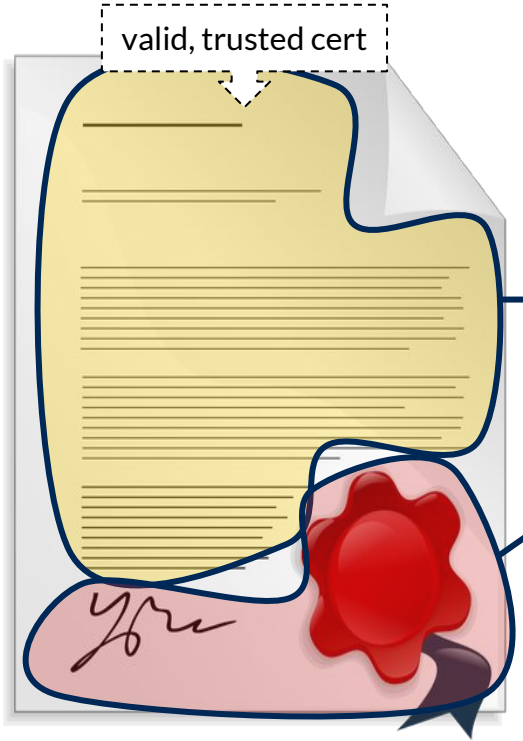

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}
```

(RFC 5280)




```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}
```

(RFC 5280)



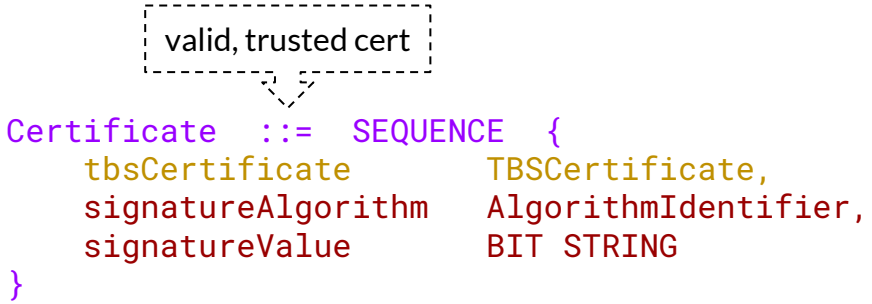

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}
```

(RFC 5280)

```
signatureValue = sign(TBSCertificate)
                 signed by CA
```

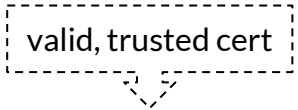
```
thumbprint = MD5(Certificate)
```

How to modify a cert without modifying it? 🤔



- `tbsCertificate` cannot be touched (it's signed)
- `signature(Algorithm|Value)` must still correctly sign
- Any room for playing games?

How to modify a cert without modifying it? 🤔




```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}

AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm    OBJECT IDENTIFIER,
    parameters   ANY DEFINED BY algorithm OPTIONAL
}
```

2.3.1 RSA Keys (RFC 3279)

The rsaEncryption OID is intended to be used in the algorithm field of a value of type AlgorithmIdentifier. The parameters field MUST have ASN.1 type NULL for this algorithm identifier.

RSA parameters trick

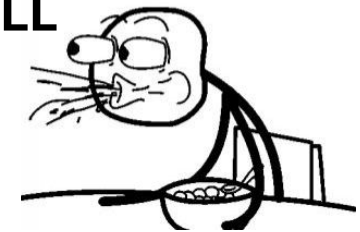
```
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm    OBJECT IDENTIFIER,
    parameters   ANY DEFINED BY algorithm OPTIONAL
}
```

- For **RSA***-signed certs, parameters must be present (but NULL)

  (* for DSA & ECDSA, parameters must be omitted) - RFC 3279

- **It turns out that implementations** don't care if it's NULL**

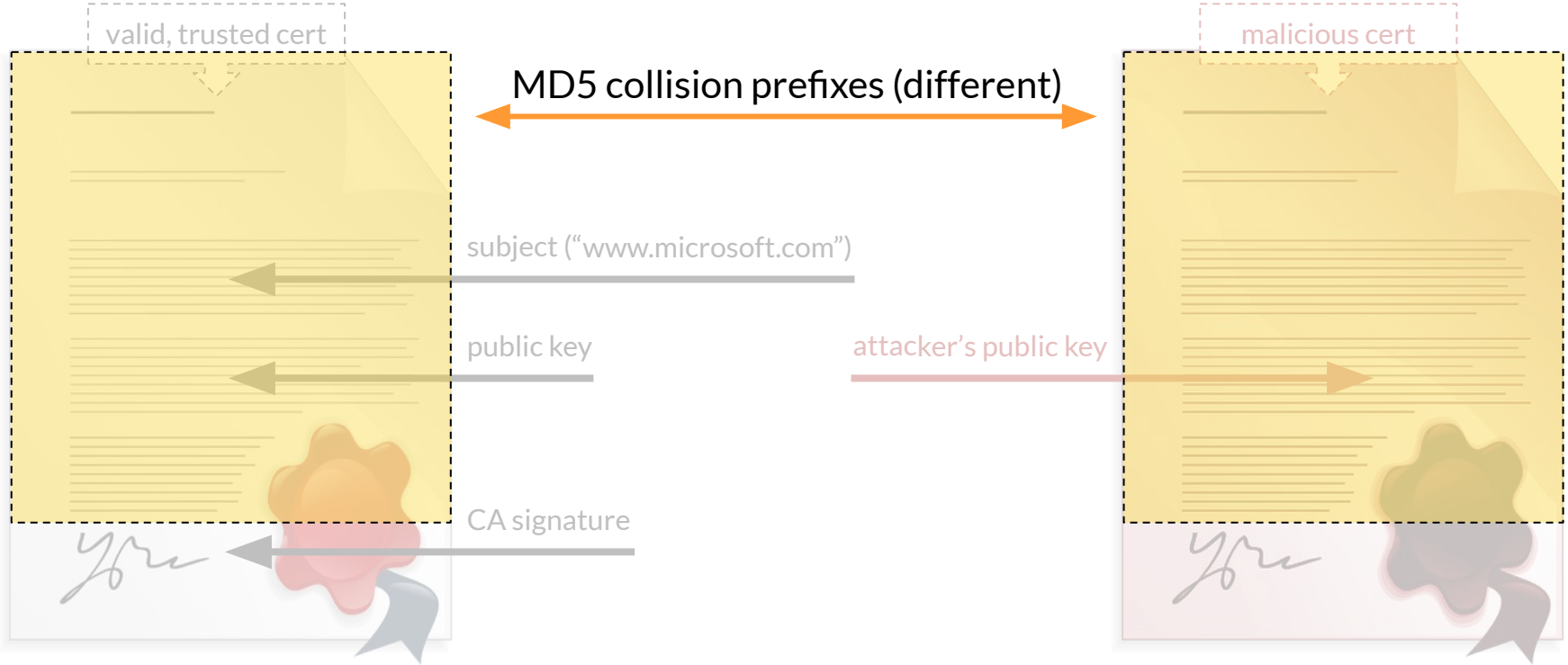
  (** CryptoAPI, OpenSSL)

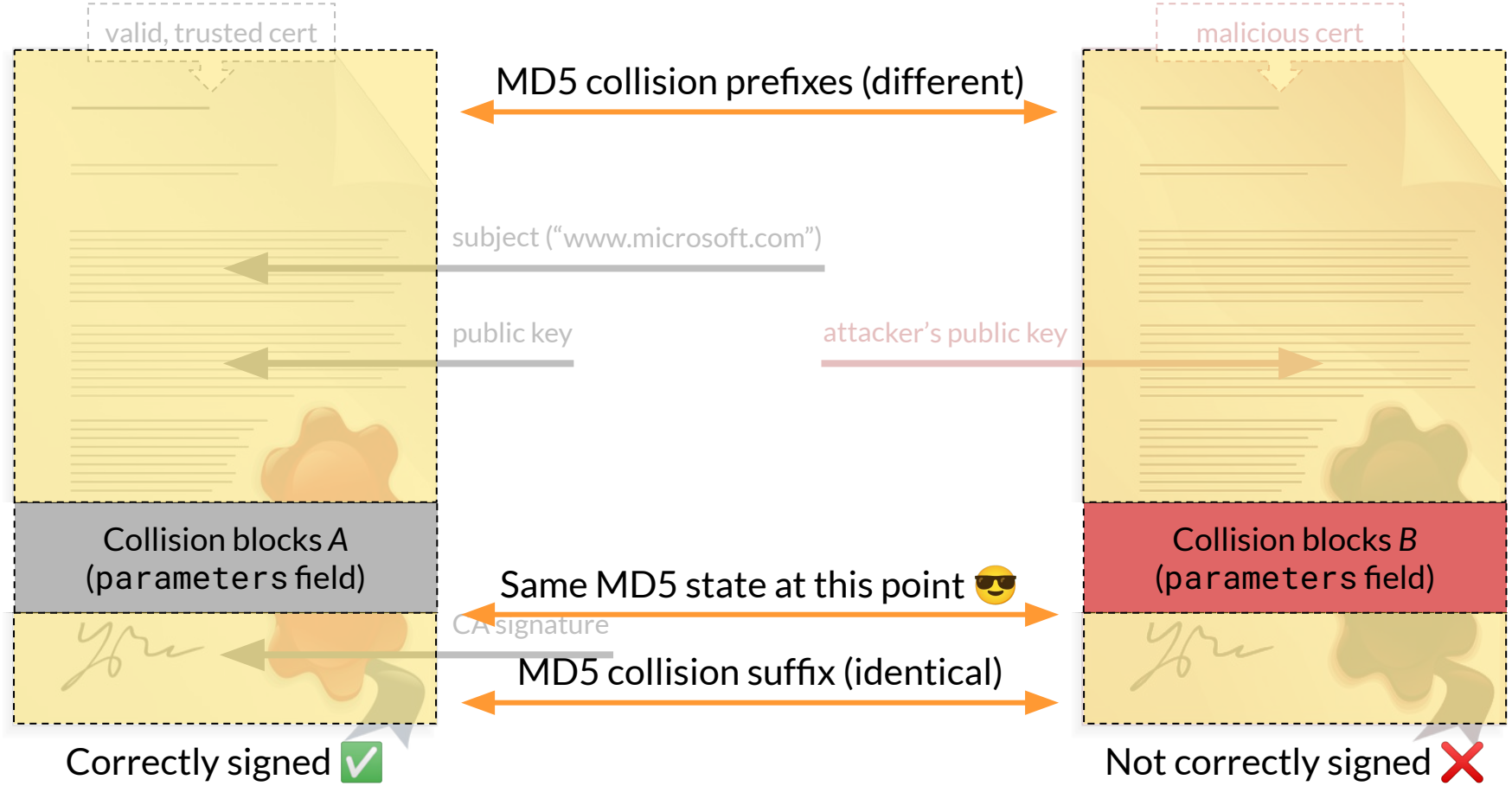


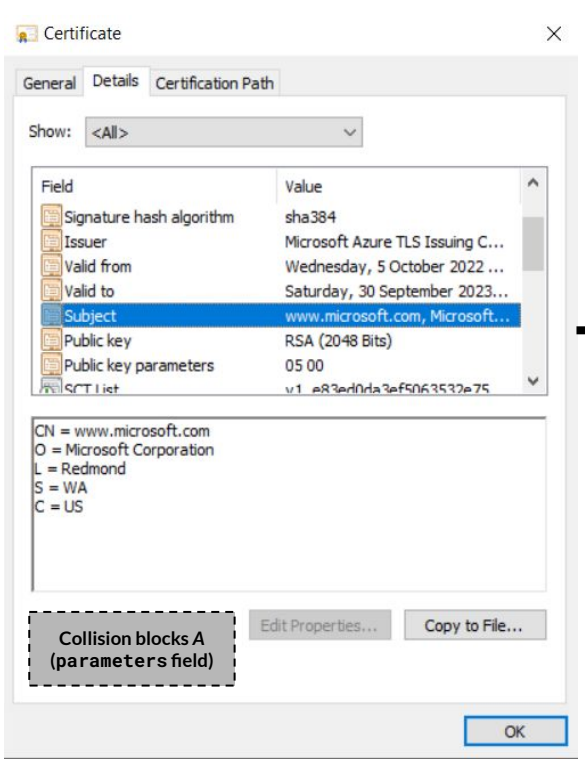
- It could be, say, a **BIT STRING** instead 🤦‍♂️

Modifying the certificate

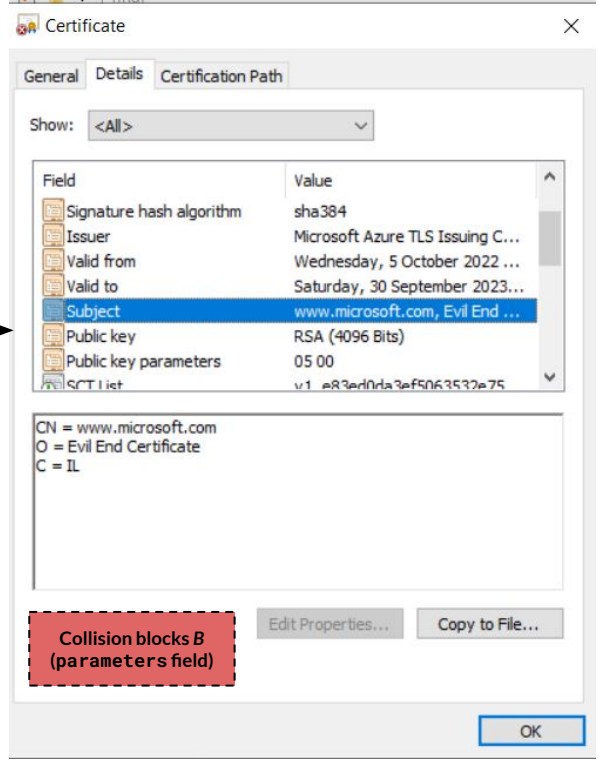
```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}

AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm    OBJECT IDENTIFIER,
    parameters   BIT STRING /* ignored - collision blocks */
}
```


valid, trusted cert
malicious cert
MD5 collision prefixes (different)
subject (“www.microsoft.com”)
public key
attacker’s public key
CA signature


valid, trusted cert
malicious cert
MD5 collision prefixes (different)
subject (“www.microsoft.com”)
public key
attacker’s public key
Collision blocks A
(parameters field)
Collision blocks B
(parameters field)
Same MD5 state at this point 😎
CA signature
MD5 collision suffix (identical)
Correctly signed ✅
Not correctly signed ❌



changed subject & public key




```
C:\TEMP\final>certutil -hashfile msft_coll.cer MD5
MD5 hash of msft_coll.cer:
bb04b2abc1972393c2778f96621d8ea2
CertUtil: -hashfile command completed successfully.
```

=

```
C:\TEMP\final>certutil -hashfile evil_coll.cer MD5
MD5 hash of evil_coll.cer:
bb04b2abc1972393c2778f96621d8ea2
CertUtil: -hashfile command completed successfully.
```

Finding a target

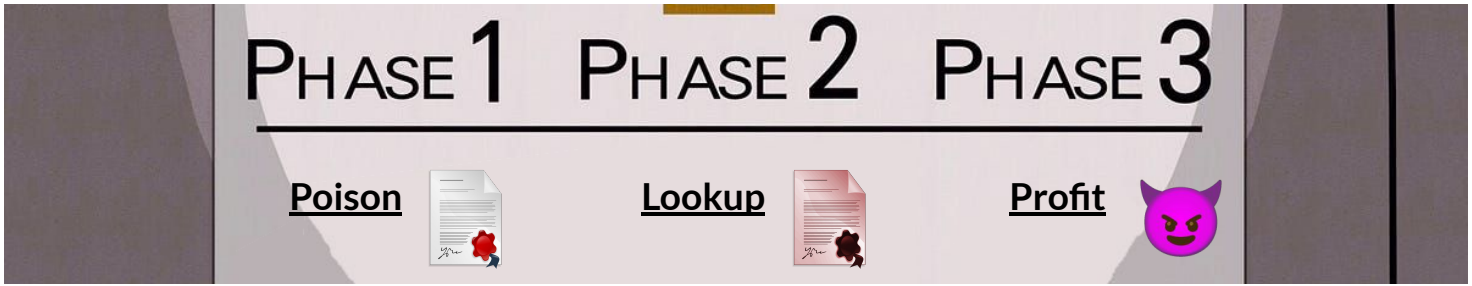
- **Locally** by yara rules and osquery we found ~200 potential targets from which 20 had a likelihood of being vulnerable

- **On GitHub** by searching for "`CERT_CHAIN_CACHE_END_CERTIFICATE`" came up with over 50 pages of code snippets containing the flag

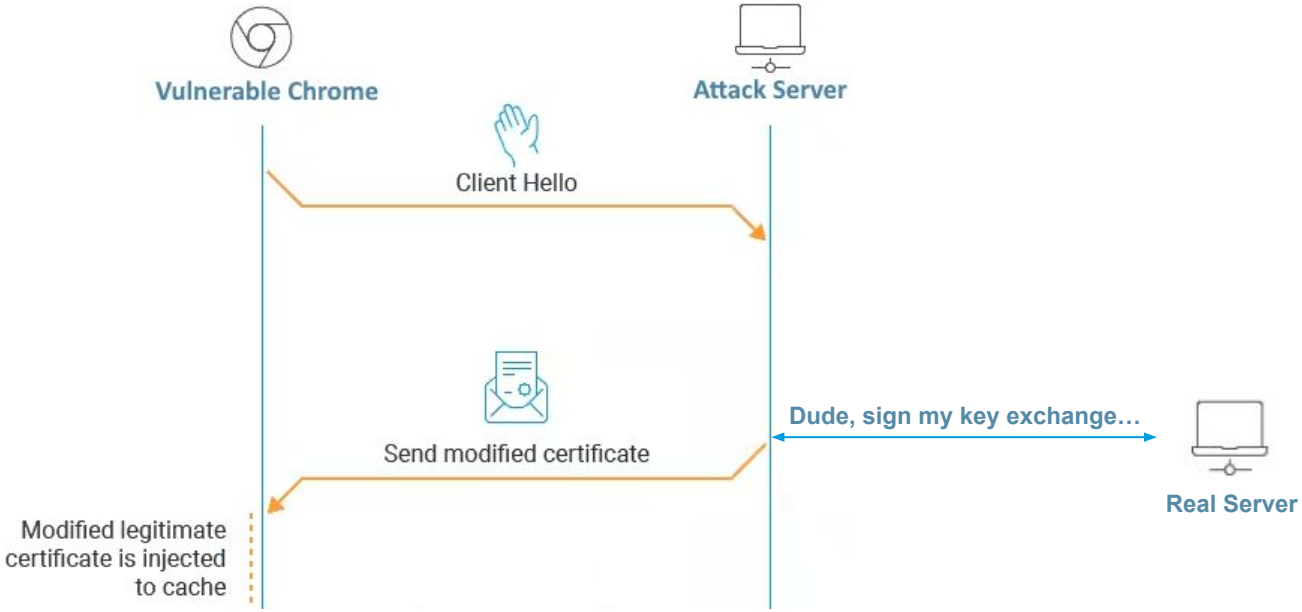
- Many candidates, not many exploitable

Target acquired: Chrome v48 (2016)

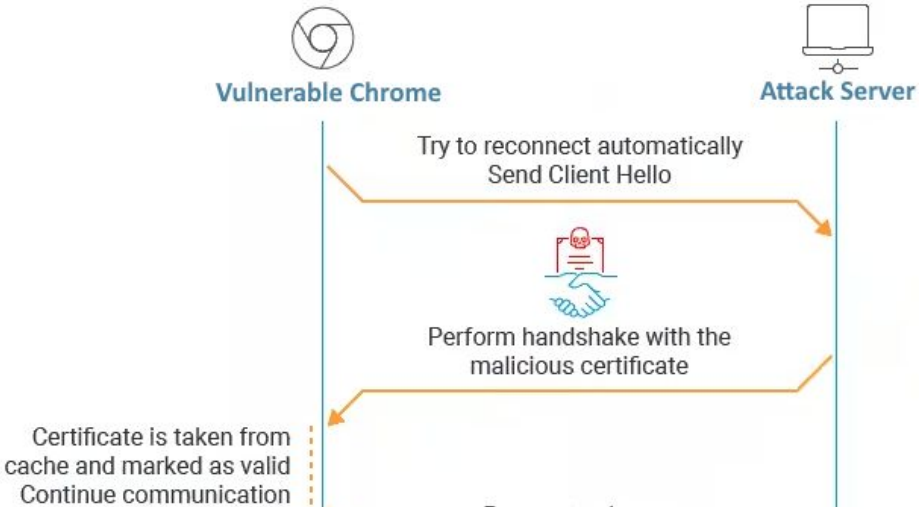
Implementing the attack on TLS



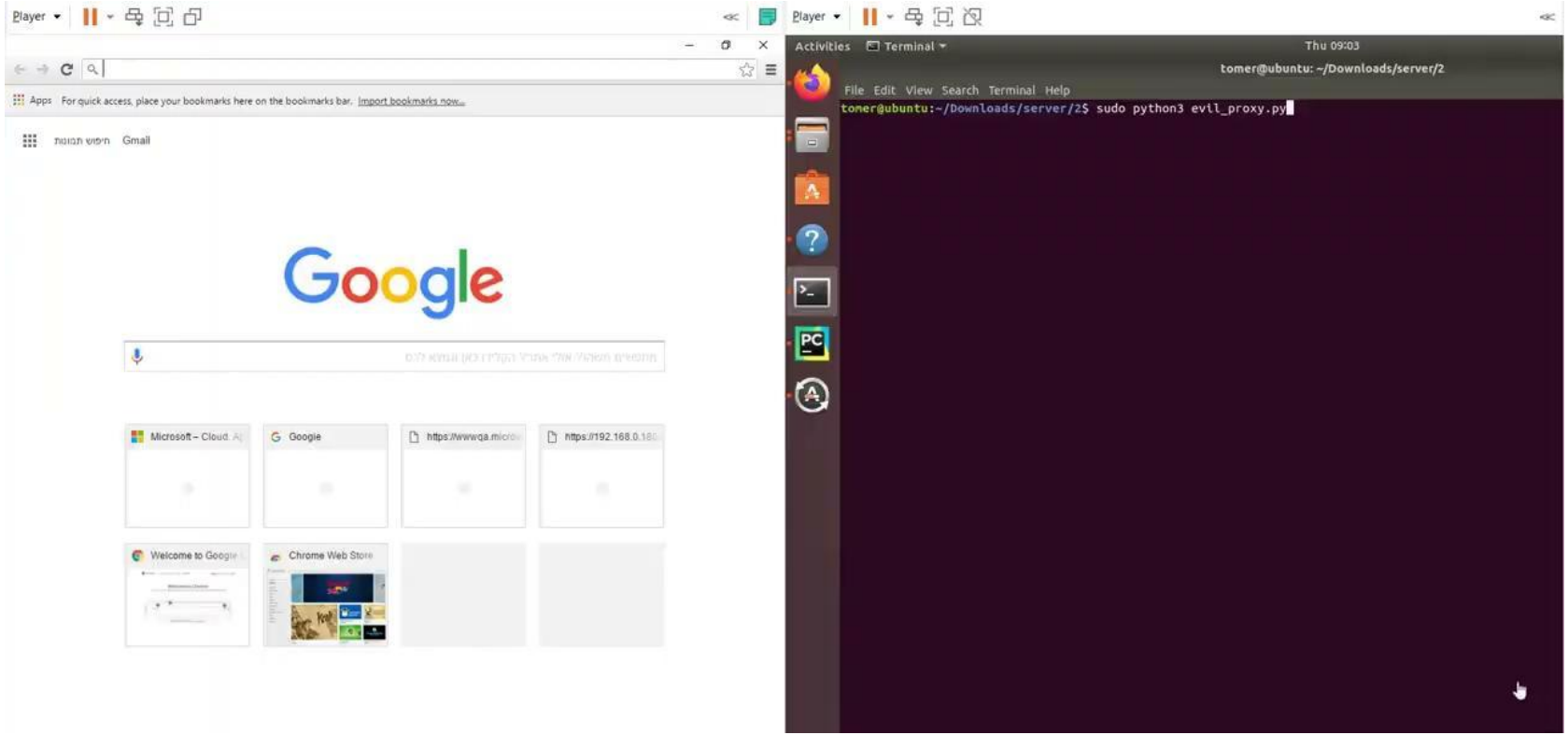
Phase 1: “Poison”



Phase 2: “Lookup”



Demo #2 - full attack



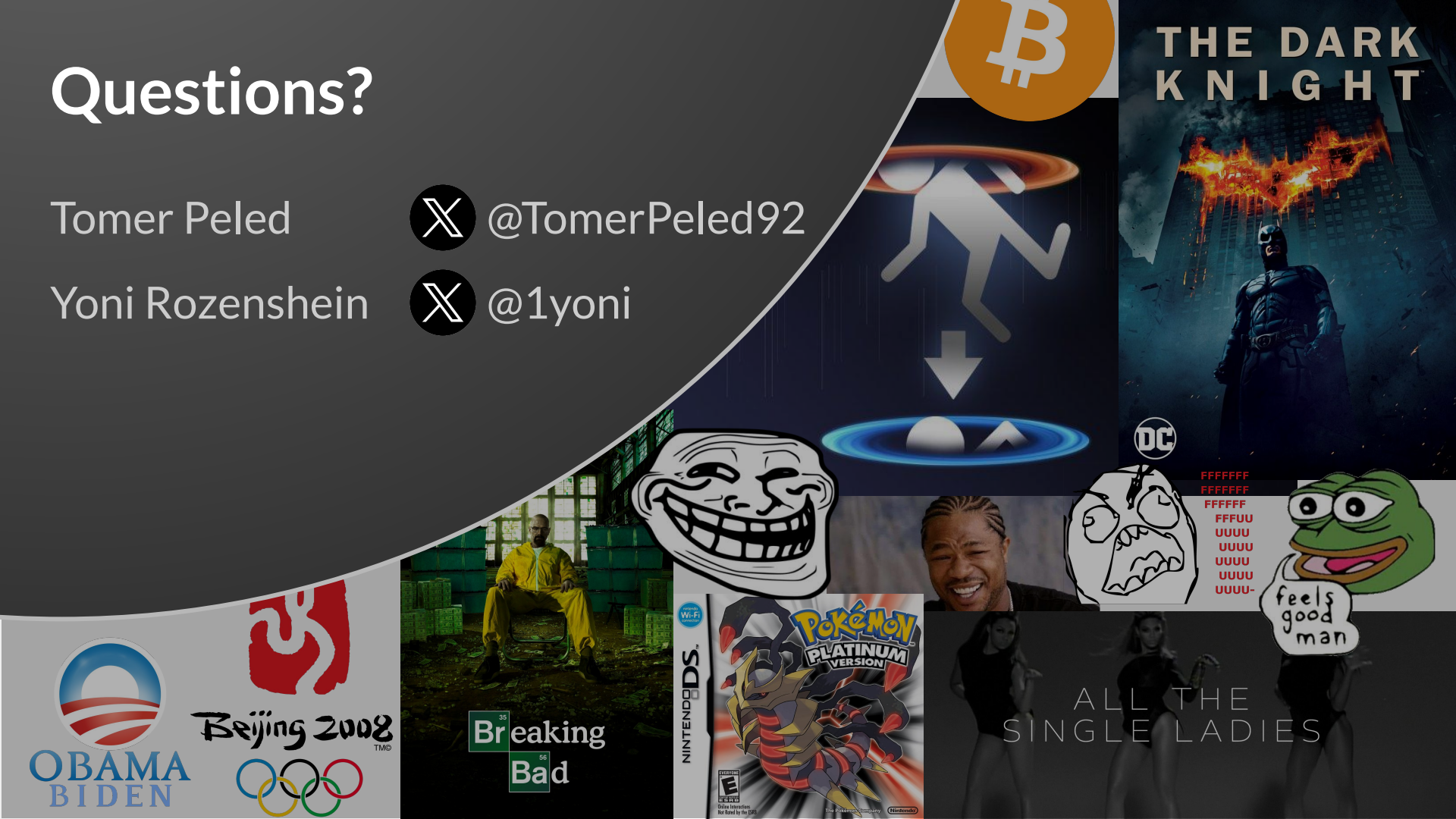
Key takeaways

- Bugs reported by NSA & GCHQ give nice *n*-days
- MD5 is still used after 2008 - even “near certificates”
- Bug class: “My hash-table hash function doesn’t have collisions”
- Don’t forget: MD5 collisions are super easy to compute!

Questions?

Tomer Peled @TomerPeled92

Yoni Rozenshein @1yoni



Questions?

Thank you!

Tomer Peled  @TomerPeled92

Yoni Rozenshein  @1yoni

References

- Blog post: https://www.akamai.com/blog/security-research/exploiting-critical-spoofing-vulnerability-microsoft-cryptoapi
- PoC: https://github.com/akamai/akamai-security-research/tree/main/PoCs/CVE-2022-34689