

Windows Into the Past: Exploiting Legacy Crypto in Modern OS Kerberos Implementation

Michal Shagam
Ronen

Eyal

Tel Aviv University



TEL AVIV אוניברסיטת
UNIVERSITY תל אביב

Our Findings

- We broke Kerberos!
- Improved the Bleichenbacher attack
- End-to-end attack on Windows
- Windows – used everywhere

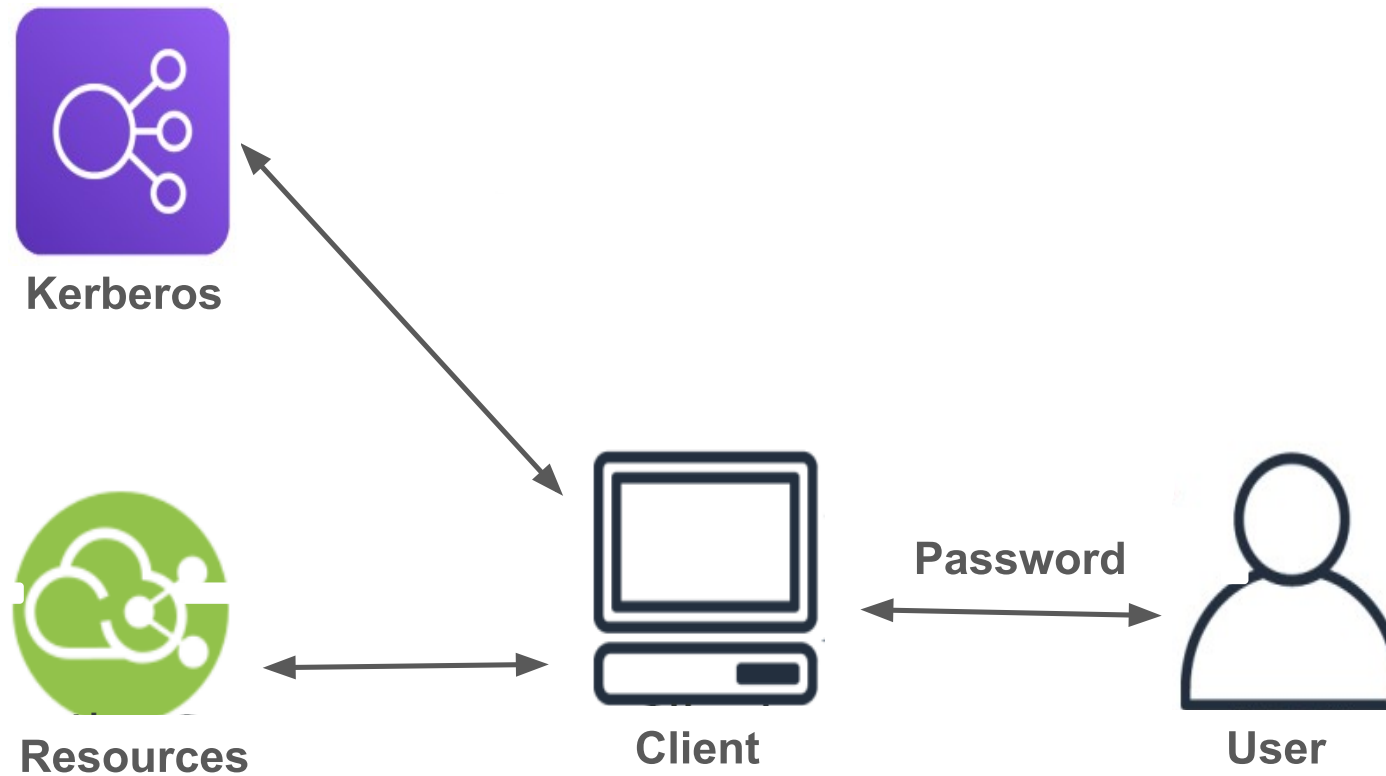


What is the Kerberos Protocol?

- Authentication protocol over non-secure networks
- Provides **Secure communication**
- Provides **Secure access to services**



Simplified Look At Kerberos

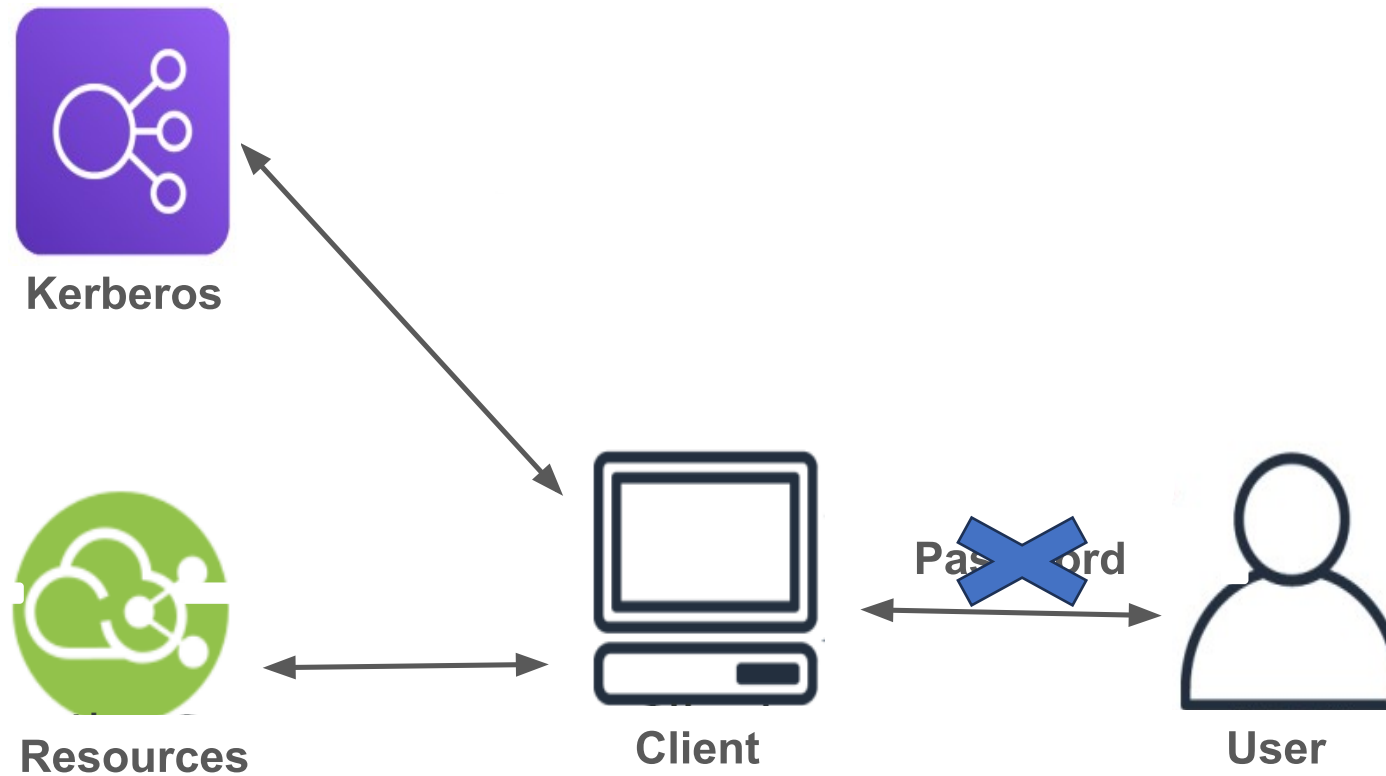


In Passwords We Trust?

- Dictionary Attacks
- Phishing
- Password DB Breaches



Simplified Look At Kerberos

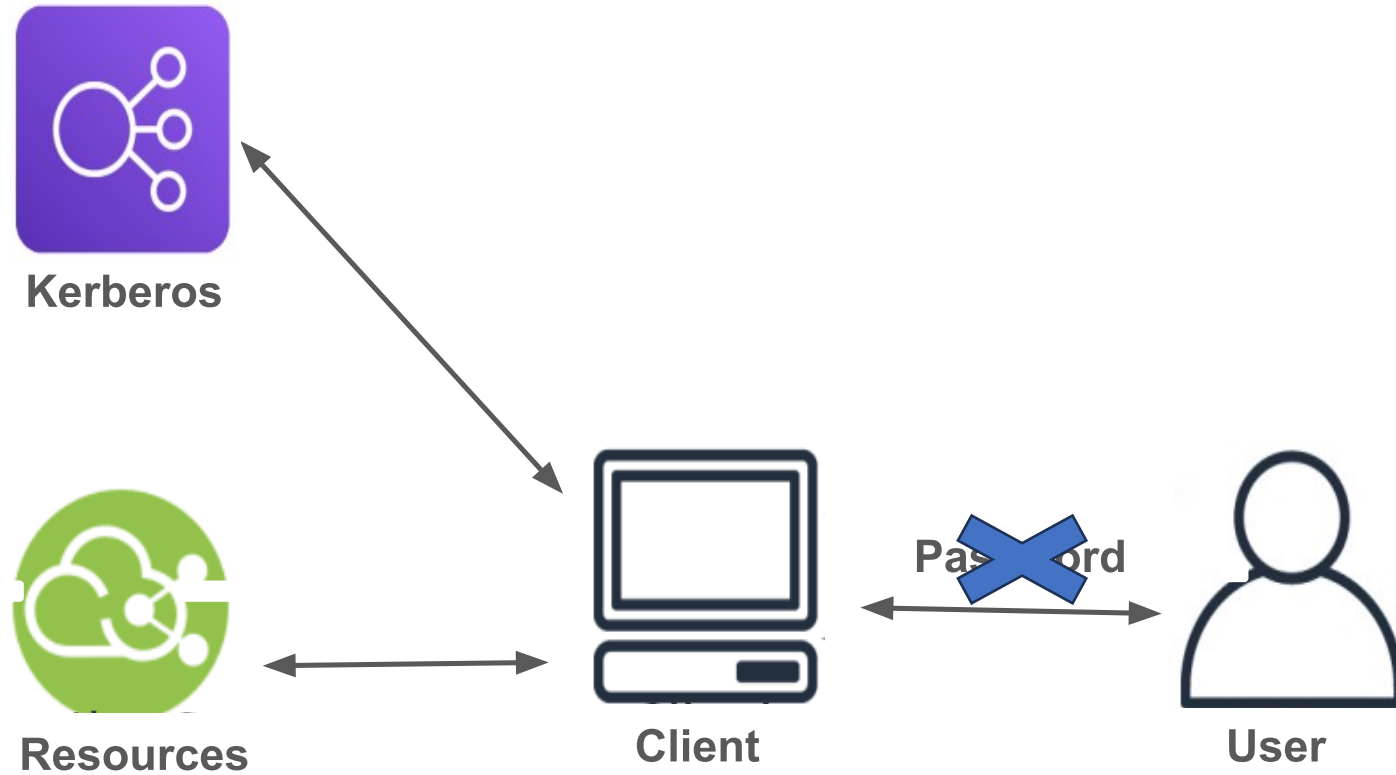


No More Passwords – Secure?

- We got rid of the password – Security!



Simplified Look At Kerberos



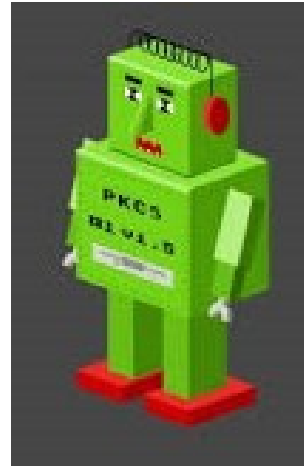
RSA + PKCS1 V1.5



Bleichenbacher Attack Returns



- Bleichenbacher Attack (1998) - PKCS#1 V1.5 vulnerable
- Manger's Attack (2001) – both PKCS#1 V1.5 and V2.0 vulnerable
- More and more attacks (SSLv2, TLS):



- No attacks on Kerberos – Why?

Possible Bleichenbacher Oracles in Kerberos

- Finding Bleichenbacher oracles in Kerberos is not trivial

- Error messages

- no error messages sent - connection times out



- Timing analysis

- connection timeout independent of RSA decryption



- Microarchitectural side channels

- RSA decryption is done on the smartcard



Layers of RSA Decryption

- Modeled in layers
 - Mathematical computation
 - Unpadding - Removal and validation of padding
 - Protocol level error handling and padding oracle mitigations
- Mathematical and data conversion runs on smartcard
- Unpadding and protocol level handling runs on Client CPU
 - This can be targeted by a microarchitectural side channel attack!



Survey of Common Kerberos Implementations

- Windows and Heimdal (used by MacOS) nonconstant time protocol implementations, no Bleichenbacher mitigations

- MIT implementation implements mitigations

- Windows and OpenSC smartcard interfaces are nonconstant time

Implementation Vulnerability Survey

OS	Kerberos Protocol	Smartcard Interface	Vulnerable
Windows	Security Package	basecsp	both
macOS	Heimdal	OpenSC	both
FreeBSD	Heimdal	OpenSC	both
Ubuntu	MIT	OpenSC	interface
RedHat	MIT	OpenSC	interface
Gentoo	MIT	OpenSC	interface
OpenSuse	MIT	OpenSC	interface
CentOS	MIT	OpenSC	interface
ArchLinux	MIT	OpenSC	interface
Suse	MIT	OpenCryptoki	no ^a

REing Windows

The screenshot displays the Immunity Debugger interface with three main panes:

- Left Pane (Decompiler):** Shows the decompiled C code for the function `CspUnpadData`. The code includes variable declarations, conditional logic, and calls to other functions like `WppTraceIndent` and `WppSF_s`.
- Middle Pane (Functions):** A table listing 46 functions from the `basecsp.dll` library. The selected function is `CspUnpadData` at location `180006330`.
- Right Pane (Listing):** Shows the assembly code for `CspUnpadData`. It starts with stack frame setup and includes instructions like `MOV`, `FUSH`, `CALL`, and `JZ`. It also shows XREFs to other functions and data.

Functions List:

Label	Location	Function Signature	Function Size
InitializeCspState	180001c4c	ulong Initialize...	288
InitializeCspCaching	18000ec9c	ulonglong Initia...	177
I_CspReadCardCacheFile	18000e3b8	ulong I_CspReadC...	316
I_CspQueryKeySizes	18000bb98	ulong I_CspQuery...	604
I_CspQueryFreeSpace	18000b86c	ulong I_CspQuery...	592
I_CspQueryCapabilities	18000a164	ulong I_CspQuery...	587
I_CspCacheInitializeQue...	18000e964	void I_CspCacheI...	81
GetCspState	180001d74	ulong GetCspStat...	82
DeleteCspState	1800019d8	ulong DeleteCspS...	621
CspWriteRootCertStore	18000d9fc	undefined CspWri...	437
CspWriteFile	18000b5d8	undefined CspWri...	367
CspVerifyCachedPinConsi...	18000ac54	undefined CspVer...	275
CspUnpadData	180006330	ulong CspUnpadDa...	353
CspRemoveCachedPin	18000ab44	undefined CspRem...	264
CspReAllocH	18001a6e0	undefined CspRea...	59
CspReadUserCertificate	18000dea0	undefined CspRea...	334
CspReadFile	18000b3ac	undefined CspRea...	546
CspQueryKeySizes	18000bdfc	ulonglong CspQue...	214
CspQueryFreeSpace	18000bac4	ulonglong CspQue...	205
CspQueryCardCacheForIt...	18000e724	ulonglong CspQue...	567
CspQueryCapabilities	18000a3b8	ulonglong CspQue...	213
CspPadData	18000bee0	ulonglong CspPad...	147
CspLeaveCriticalSection	18001a7b8	undefined CspLea...	87
CspInitializeCriticalSection	18001a728	undefined CspIn...	26
CspIncrementCacheFresh...	18000e4fc	ulonglong CspInc...	263
CspImportsSimpleBlobHelper	180005ba4	ulong CspImports...	292
CspGetContainerInfo	18000a684	undefined CspGet...	1126
CspEnterCriticalSection	18001a750	undefined CspEm...	92
CspEndTransaction	18000c9c	ulonglong CspEnd...	168
CspDeleteFile	18000b750	undefined CspDel...	275
CspDeleteContainer	18000a494	undefined CspDel...	264
CspCreateFile	18000b2d4	ulonglong CspCre...	208
CspCreateContainer	18000a5a4	ulonglong CspCre...	214
CspCheckContainerMapRe...	18000c734	ulonglong CspChe...	190
CspCacheLookupFileProc	18000ea0	ulong CspCacheLo...	287
CspCacheDeleteFileProc	18000ec20	ulong CspCacheDe...	114
CspCacheAddFileProc	18000e9c0	ulong CspCacheAd...	286
CspBuildUserCertStore	18000d500	ulonglong CspBui...	1267
CspBuildRootCertStore	18000d2ec	ulonglong CspBui...	524
CspBuildCertificateFile...	18000dbb8	undefined CspBui...	737
CspBeginTransaction	18000b7c	ulonglong CspBeg...	279
CspAuthenticateUser	180002a64	ulong CspAuthent...	427
CspAuthenticatePin	18000ad70	undefined CspAut...	726
CspApplyPKCSISigningFor...	18001a818	undefined CspAp...	652
CspAllocH	1800015e0	void * CspAllocH...	44
CspAddCardCacheItem	18000e60c	ulonglong CspAdd...	270

Assembly Listing:

```

undefined4 Stack[-0x38]:4 local_38 XREF[1]: 1800063e4(k)
?CspUnpadData@YAKFEAU_CARD_RSA_DECRYPTI_INF0@... XREF[5]: 180006475(w)
CspUnpadData FindCardBySerialNumber:1
LocalAcquireContext:1800
180023ab3(*)
180006330 48 89 5c MOV qword ptr [RSP + local_res10],RBX
24 10
180006335 55 FUSH RBP
180006336 56 FUSH RSI
180006337 57 FUSH RDI
180006338 41 56 FUSH R14
18000633a 41 57 FUSH R15
18000633c 48 83 ec 30 SUB RSP,0x30
180006340 33 ff XOR EDI,EDI
180006342 4c 8b fa MOV R15,param_2
180006345 33 d2 XOR param_2,param_2
180006347 89 7c 24 60 MOV dword ptr [RSP + local_res8],EDI
18000634b 8b f7 MOV ESI,EDI
18000634d 49 89 7c MOV qword ptr [RSP + local_res20],RDI
24 78
180006352 4d 8b 20 MOV R14,param_3
180006355 48 8b d9 MOV RBX,param_1
180006358 e9 db ae CALL WppTraceIndent void WppTraceIndent(
ff ff
18000635d 48 8b 0d MOV param_1,qword ptr [-WPP_GLOBAL_Control] = 18002c0f8
94 5d 02 00
180006364 48 8d 2d LEA RBP,[WPP_GLOBAL_Control] = 18002c0f8
8d 5d 02 00
18000636b 48 3b cd CMP param_1,RBP
18000636e 74 26 JZ LAB_180006396
180006370 f6 41 1c 02 TEST byte ptr [param_1 + 0xc]>=DAT_18002c114,0x2
180006374 74 20 JZ LAB_180006396
180006376 80 79 19 05 CMP byte ptr [param_1 + 0x19]>=DAT_18002c111,0x5
18000637a 72 1a JC LAB_180006396
18000637c 4c 8b 0d MOV R5,qword ptr [WPP_pszIndent] = NaP
a5 6b 02 00
180006383 8d 57 3a LEA param_2,[RDI + 0x3a]
180006386 48 8b 49 10 MOV param_1,qword ptr [param_1 + 0x10]>=DAT_18002c... = 0000000300000000h
18000638a 4c 8d 05 LEA param_3,[WPP_e8978bfc35ec3505198246492c009beb... = FCh
2f dd 01 00
180006391 e9 2a b0 CALL WPP_SF_s undefined WPP_SF_s(u
ff ff
LAB_180006396 XREF[3]: 18000636e(j), 180006374,
18000637a(j)
180006396 48 85 db TEST RBX,RBX
180006399 0f 84 8b JZ LAB_18000642a
00 00 00
18000639f 4d 85 ff TEST R15,R15
1800063a2 0f 84 82 JZ LAB_18000642a
00 00 00
1800063a8 4d 85 fe TEST R14,R14
1800063ab 74 7d JZ LAB_18000642a
1800063ad 83 3b 02 CMP dword ptr [RBX],0x2
1800063b0 74 0a JZ LAB_1800063bc
1800063b2 bb 1a 05 MOV EBX,0x51a
00 00 00
1800063b7 e9 80 00 JMP LAB_18000643c
  
```

Windows Non Constant-time Protocol Code

```
1 int CPImportKey(csInfo) {  
2     ...  
3     int result = CspUnpadData(csInfo, &len, &out);  
4     if (result == SUCCESS) {  
5         ...  
6         int isLegal = FIsLegalKeySize(len, out);  
7         if (isLegal) {  
8             Decrypt_3DES(encData, out, len, &decData);  
9             ...  
10        }  
11    }  
12    return SUCCESS;  
13 }
```

Our Target

- Nonconstant-time code in unpadding layer
- Flush+Reload cache attack
- Very noisy



Bleichenbacher Attack – What Do We Need?

- Padding Oracle with feedback
- Ability to modify the ciphertext


- Large number of queries
- Large number of queries and **FAST** – our deadline is 10 hours!

How Do We Generate Messages?



PIN needed



PIN needed 



PIN needed 

Harmless Looking Malicious HTML



- Browsers can access remote files!

- Stealth!

- Repetition!

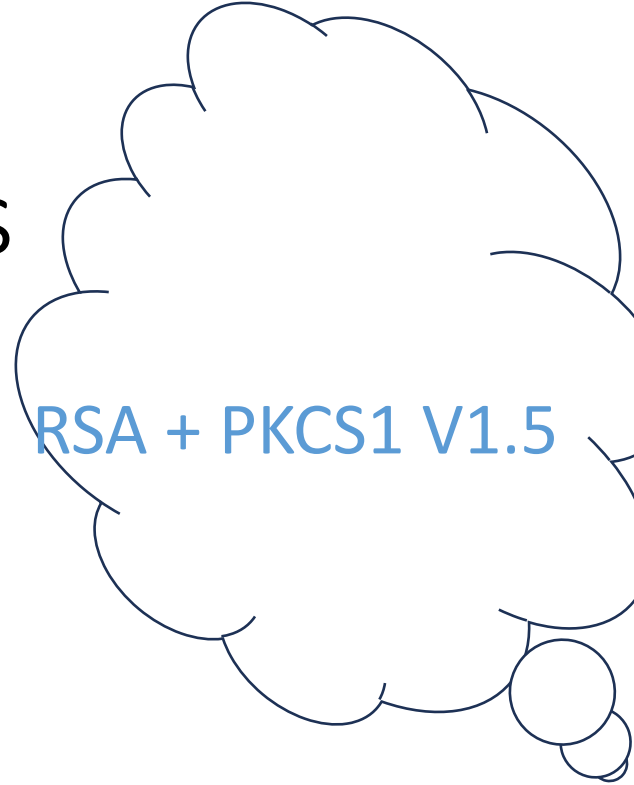
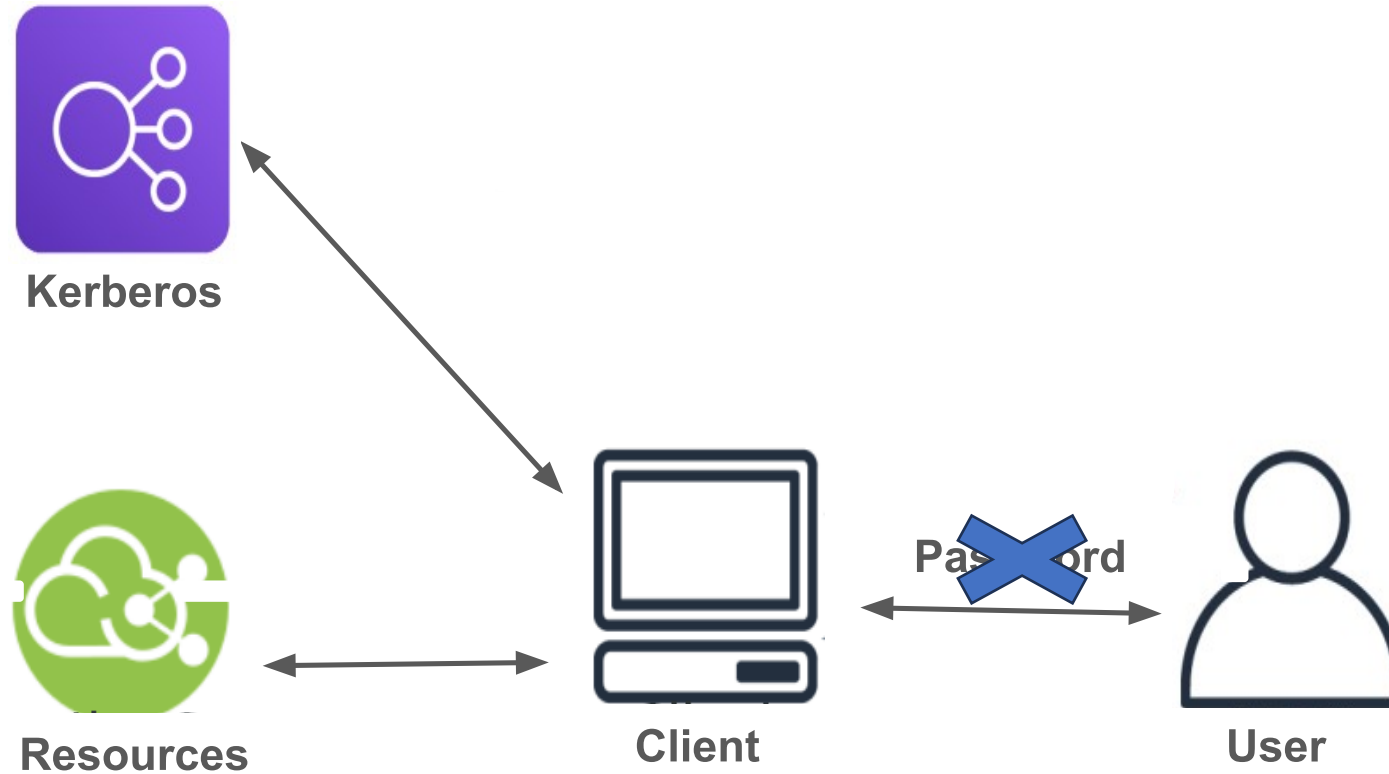
```
1 <html>
2   <head>
3     <meta http-equiv="refresh" content="4">
4   </head>
5   <iframe src="file:///win-r3f0hi0ntca\mysecret\
6     classifiedsecrets.txt">
7 </iframe>
7 </html>
```

Stealth Server Side

- Intercept and replace RSA message
- Server can detect the large volume of authentication requests
- Unauthenticated Data in Kerberos messages
- Response packet can be completely spoofed!
- We can create a fast and stealthy attack (no need for the KDC)



Accelerated+Remote Attack Kerberos



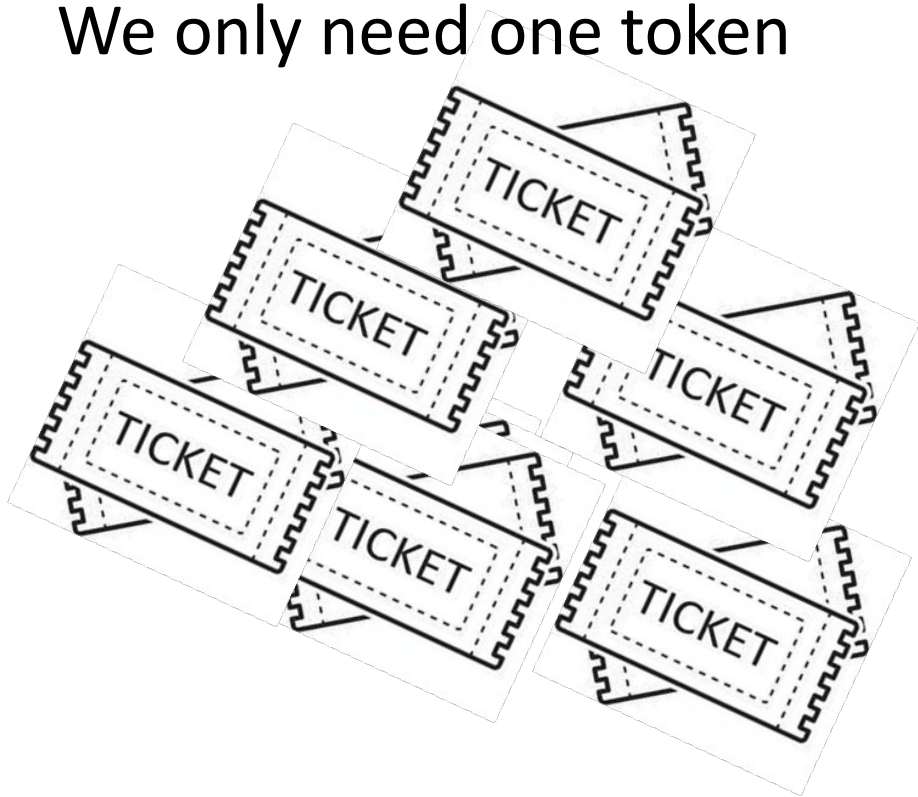
Fast Enough?

- Short answer -> sometimes for some messages
- Real tokens (40k dataset generated by the KDC)
 - ~23% decrypted in less than 20K queries
 - ~8.5% decrypted in less than 10K queries



Impersonation

- Decrypted tokens give the attacker access to the user/admin's resources
- We only need one token



Bleichenbacher's Attack (1998)

- Padding structure constraint



- Initial multiplier search



- Range reduction - Additional Multipliers



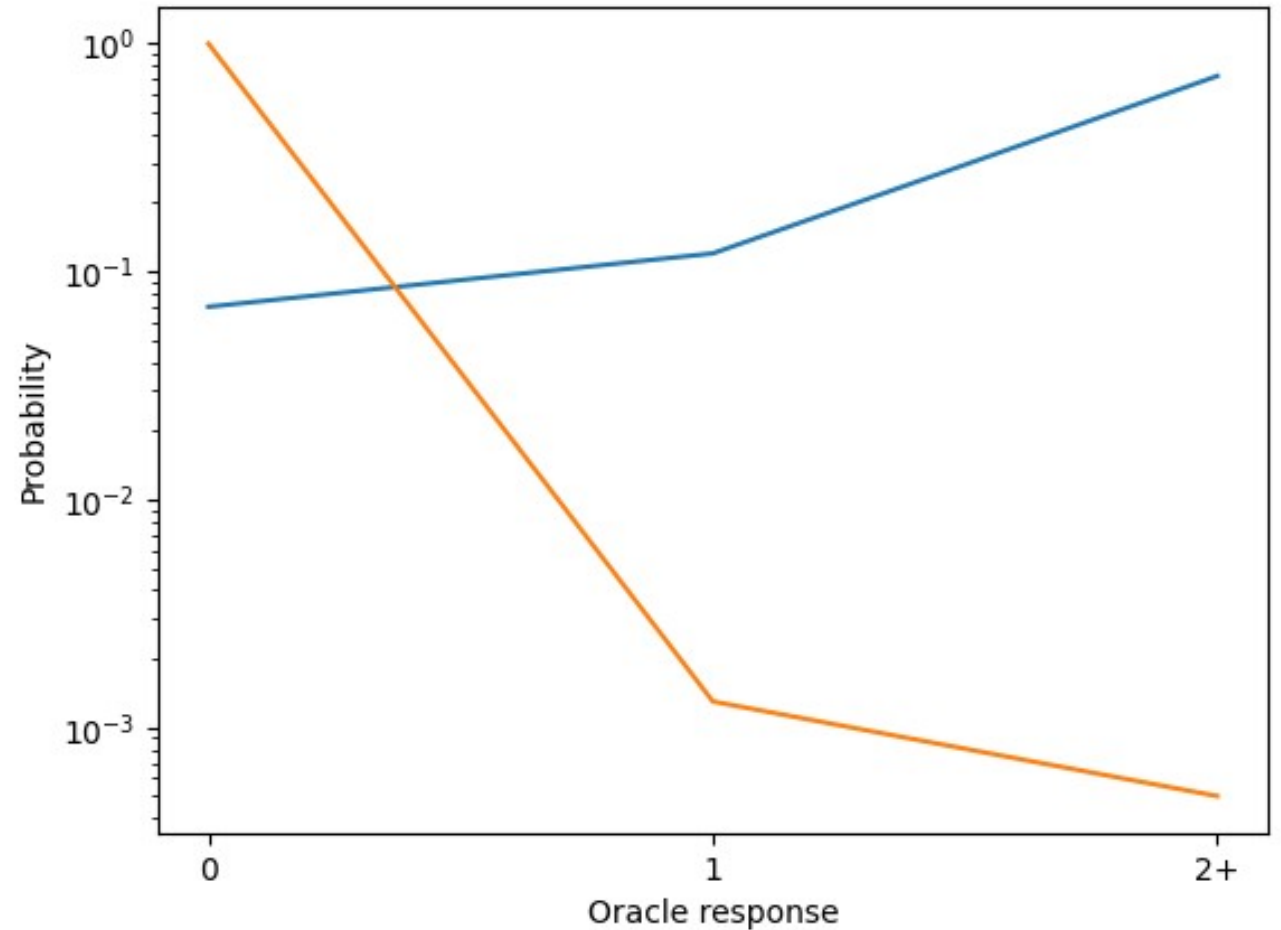
Noisy Oracle

- Oracles are often noisy!
- FN and FP have different impacts
- Repeat all queries? Inefficient
- Attack can handle FN -> aim for lower FP probability
- Repetition of positives
 - Double – repeat positive queries twice - require both be positive
 - Majority – repeat positive queries up to 3 times – require majority be positive
- Can we improve this?



Noisy Oracle

- Our oracle isn't binary!
 - Number of hits in our F+R attack matter!
- FP usually have just one hit
- Tailored query repetition algorithm



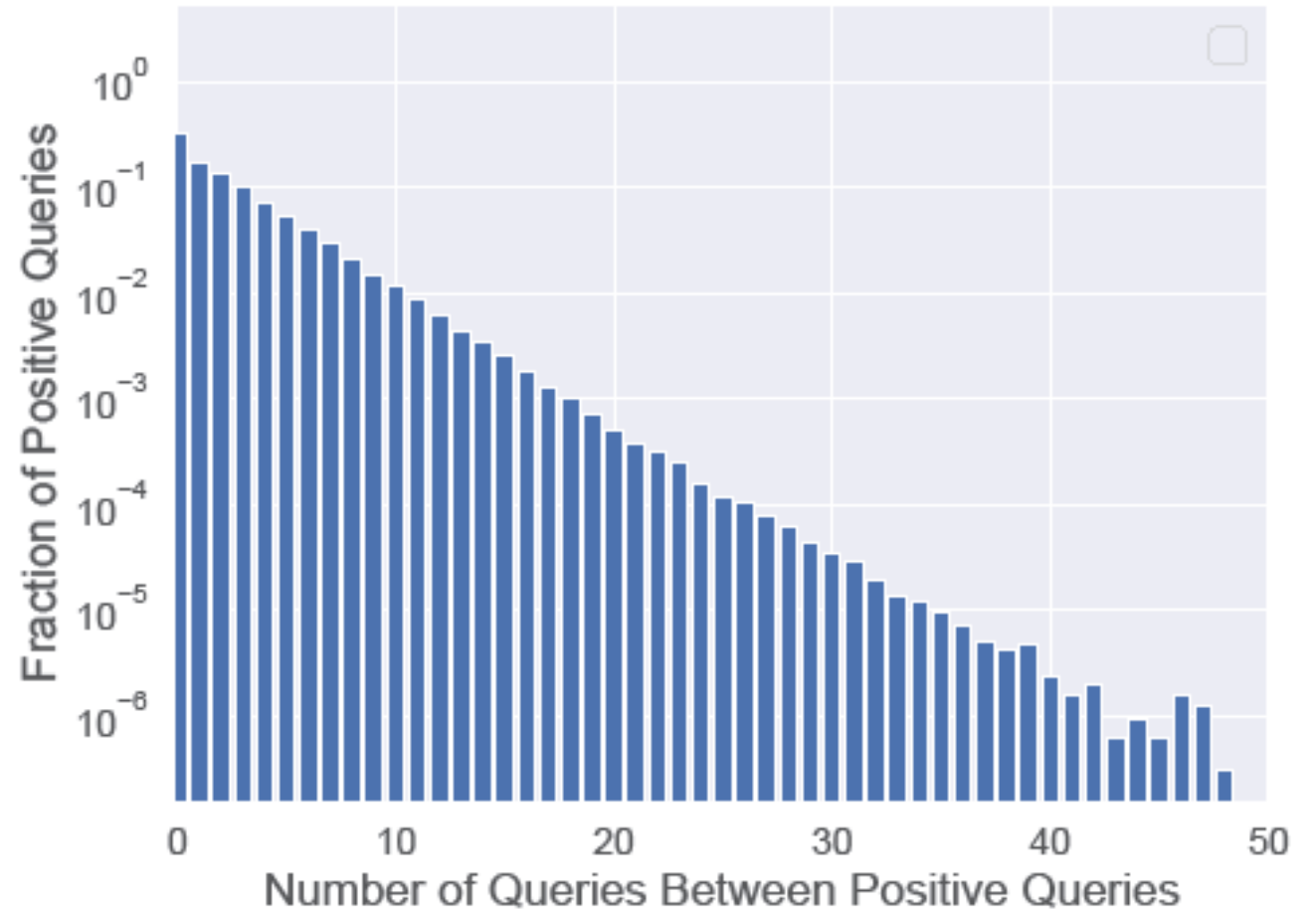
Accelerated Attack- Experiment Results

- Focus on “Fast subset” - 18% of messages
- Attack performed successfully on 15 of 17 “fast” tokens
- We still have a nonzero probability for a FP



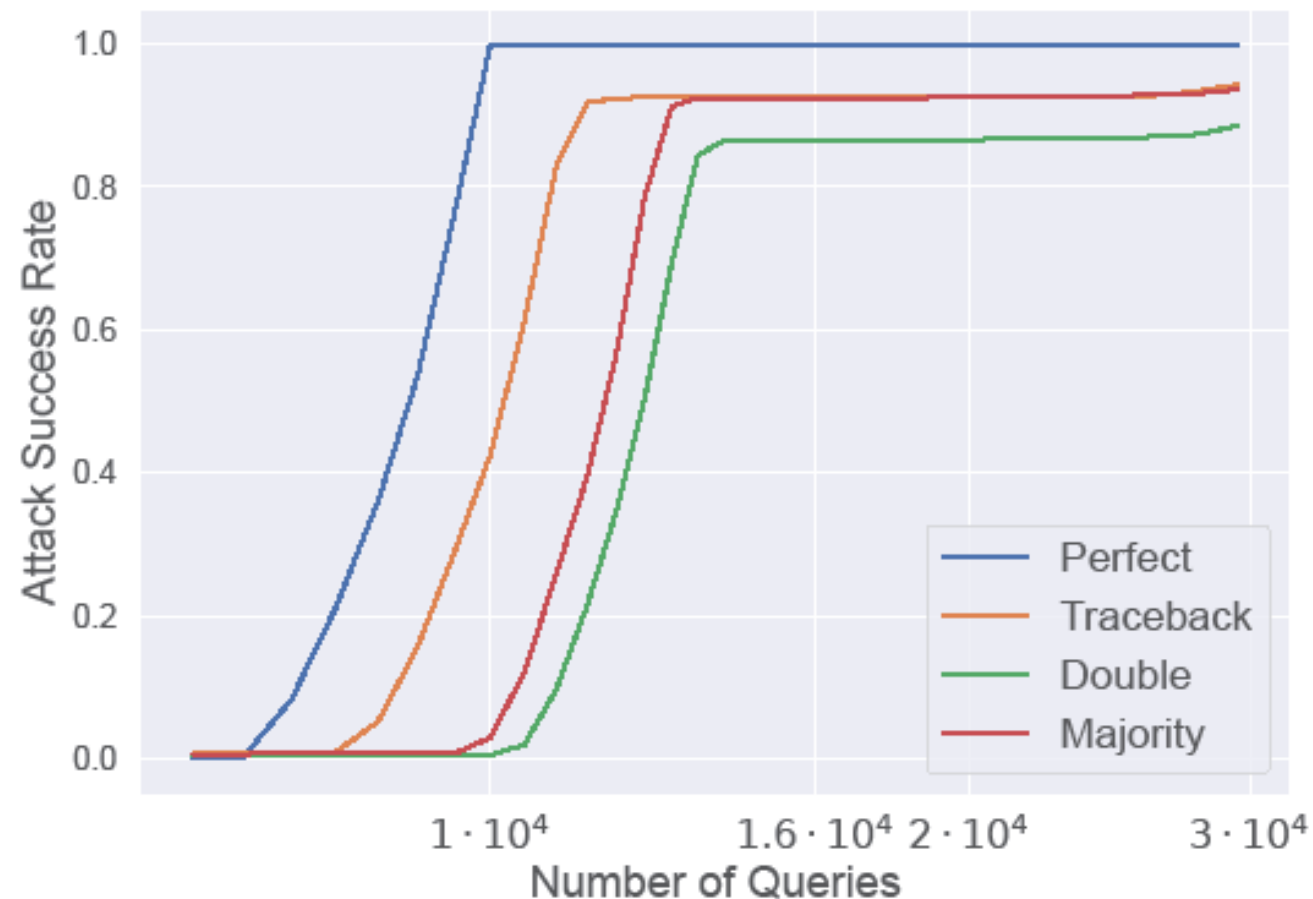
False Positive Detection

- Upper bound for amount of false queries
- Too many false queries in a row
-> FP
- State has changed
- Traceback!



Traceback – Attack Recovery

- Save previous states
- Count false queries
- Revert to a previous state if FP is detected
- Traceback method much closer to the perfect oracle



Who Is Fast?

- Correlation between attack steps
- Fast initial search -> Fast range reduction
- Message classification
- Detect early and abort!



Early Abort – Experiment Results

- Experiment time: 3 days
- Detection limit: 600 queries
- Continue attack iff initial multiplier search is completed

#	n_{msgs}	T_q	T [hours]	$Q_{<16k}$
1	36	29336	18.4	8309
2	30	26185	16.4	8680
3	29	27023	16.9	9367

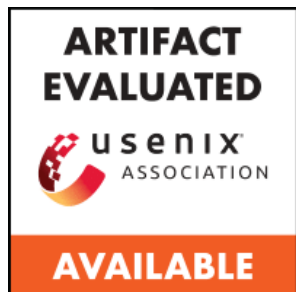
Ghost of Crypto Past

- Nonconstant time implementations -> side channel attacks
- Lack of Forward Secrecy -> compromise other protocols
- Unlimited RSA decryption attempts – why?



Recap + Questions

- Broke Kerberos authentication protocol
- Improved Bleichenbacher Attack Methodology
- End-to-End impersonation attack on Windows
- One PIN code entry, one site, one day -> control!



Git Link

