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

Michal Shagam Eyal Ronen

Tel Aviv University

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

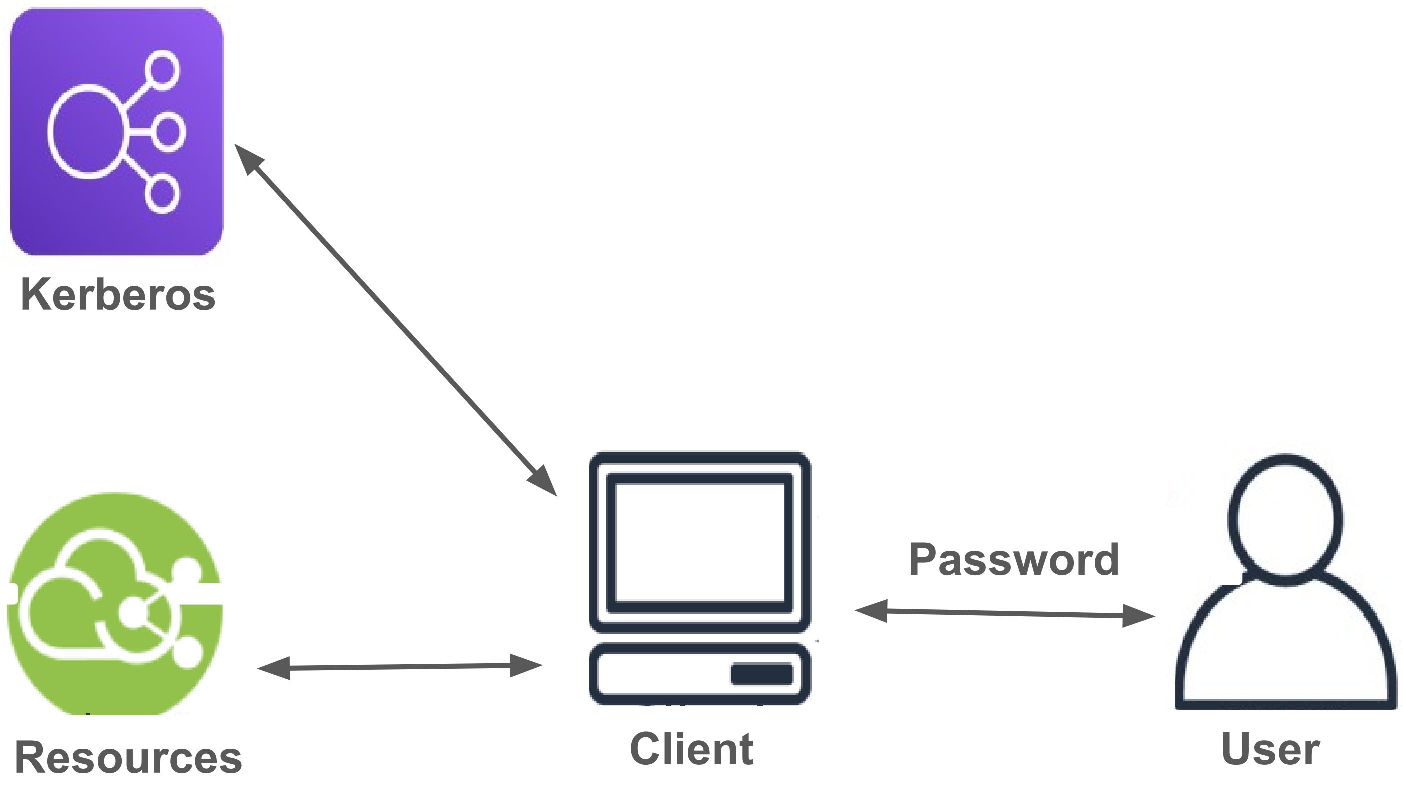
Our Findings

- 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

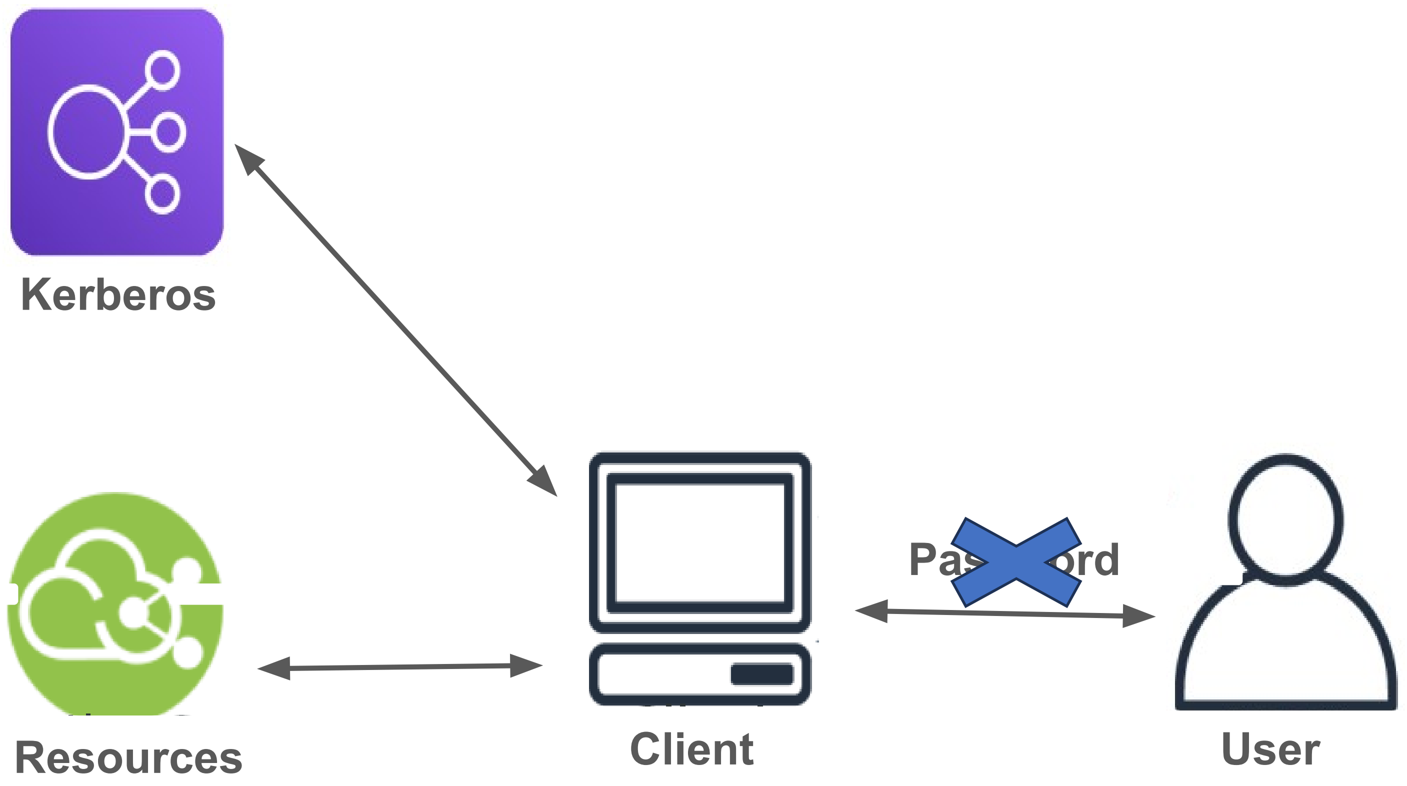
Simplified Look At Kerberos



In Passwords We Trust?

- Dictionary Attacks
- Phishing
- Password DB Breaches

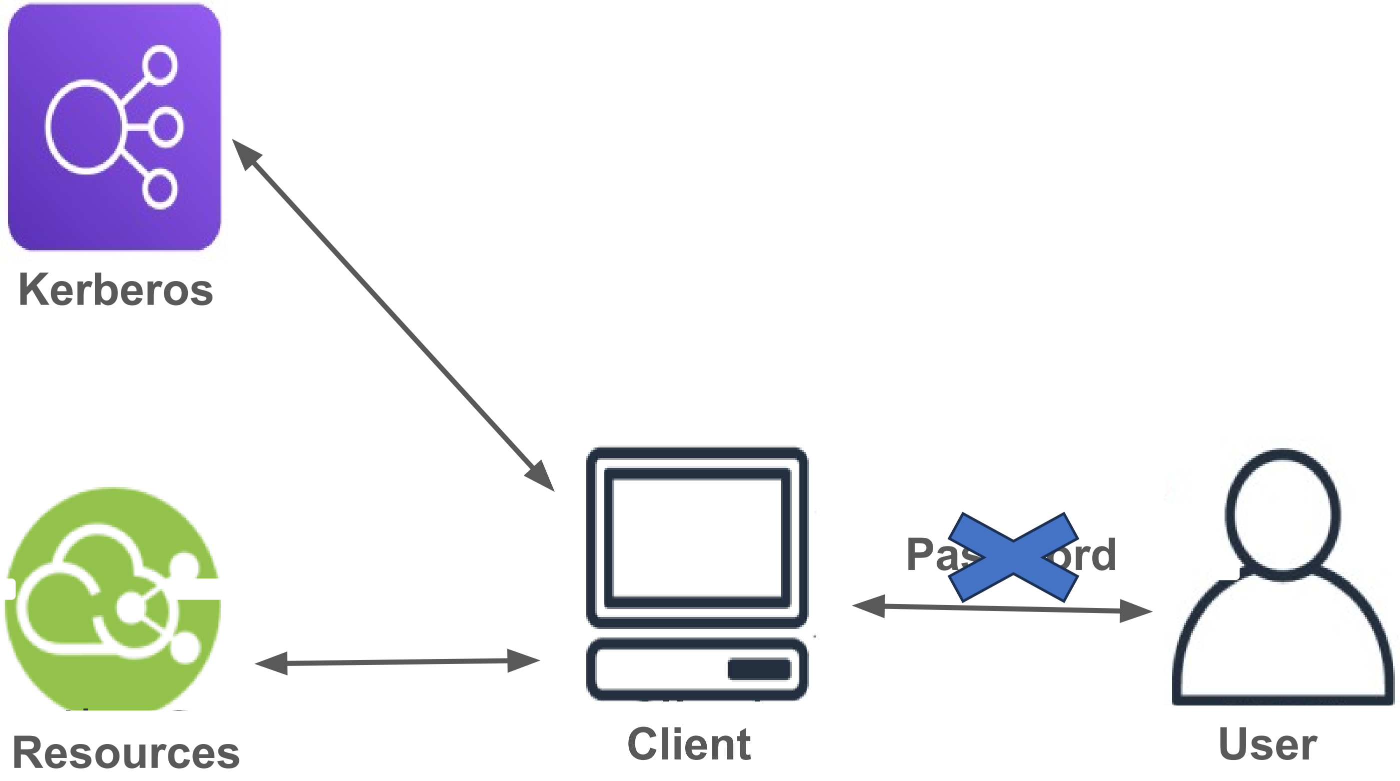
Simplified Look At Kerberos

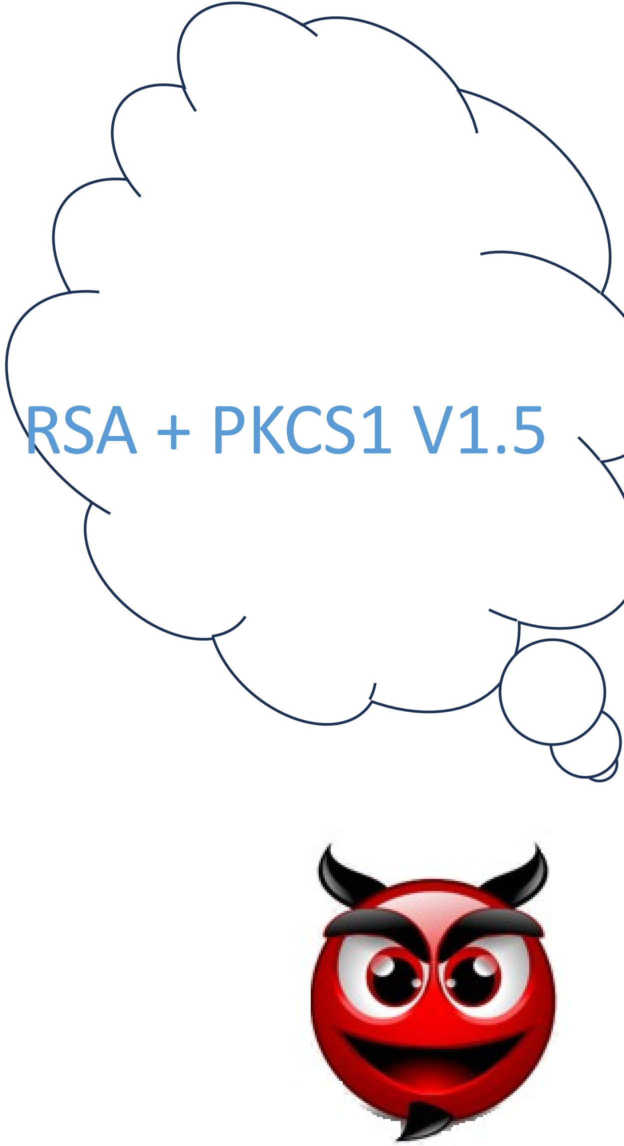


No More Passwords – Secure?

- We got rid of the password – Security!

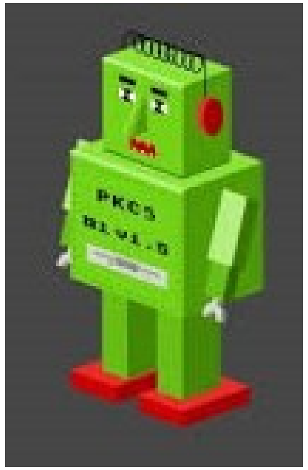
Simplified Look At Kerberos

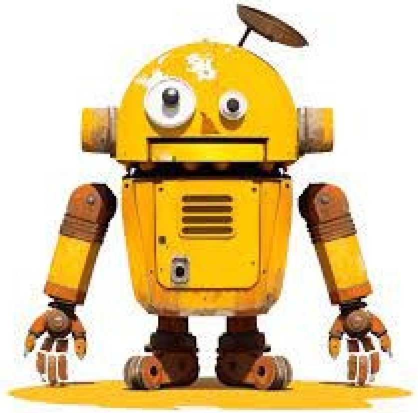




Bleichenbacher Attack Returns

- 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!

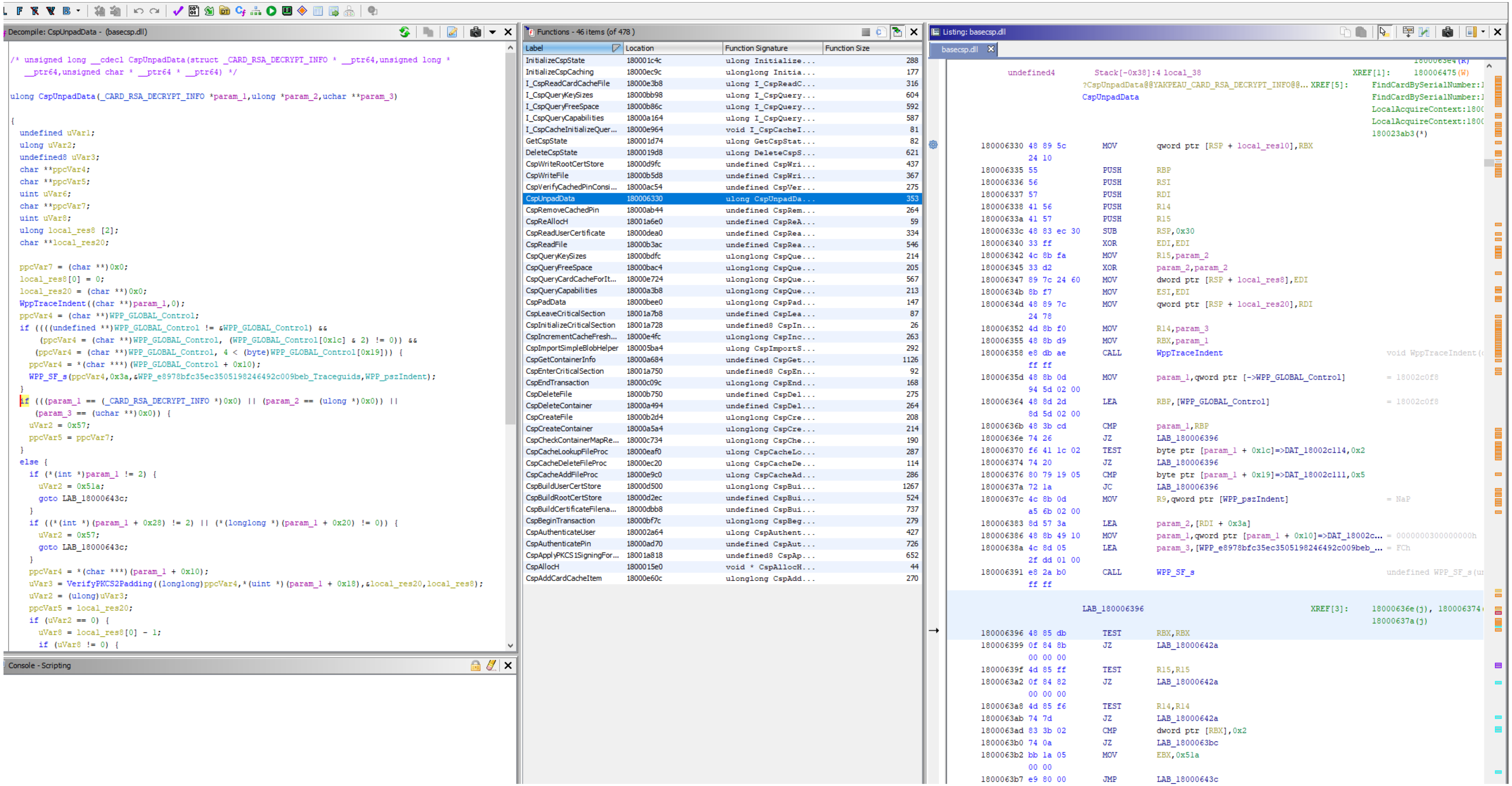
Survey of Common Kerberos Implementations

- 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



Windows Non Constant-time Protocol Code

```
int CPImportKey(csInfo){
  ...
  int result = CspUnpadData(csInfo, &len, &out);
  if (result == SUCCESS){
    ...
    int isLegal = FIsLegalKeySize(len, out);
    if (isLegal){
      Decrypt_3DES(encData, out, len, &decData);
      ...
    }
  }
  return SUCCESS;
}
```

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!

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

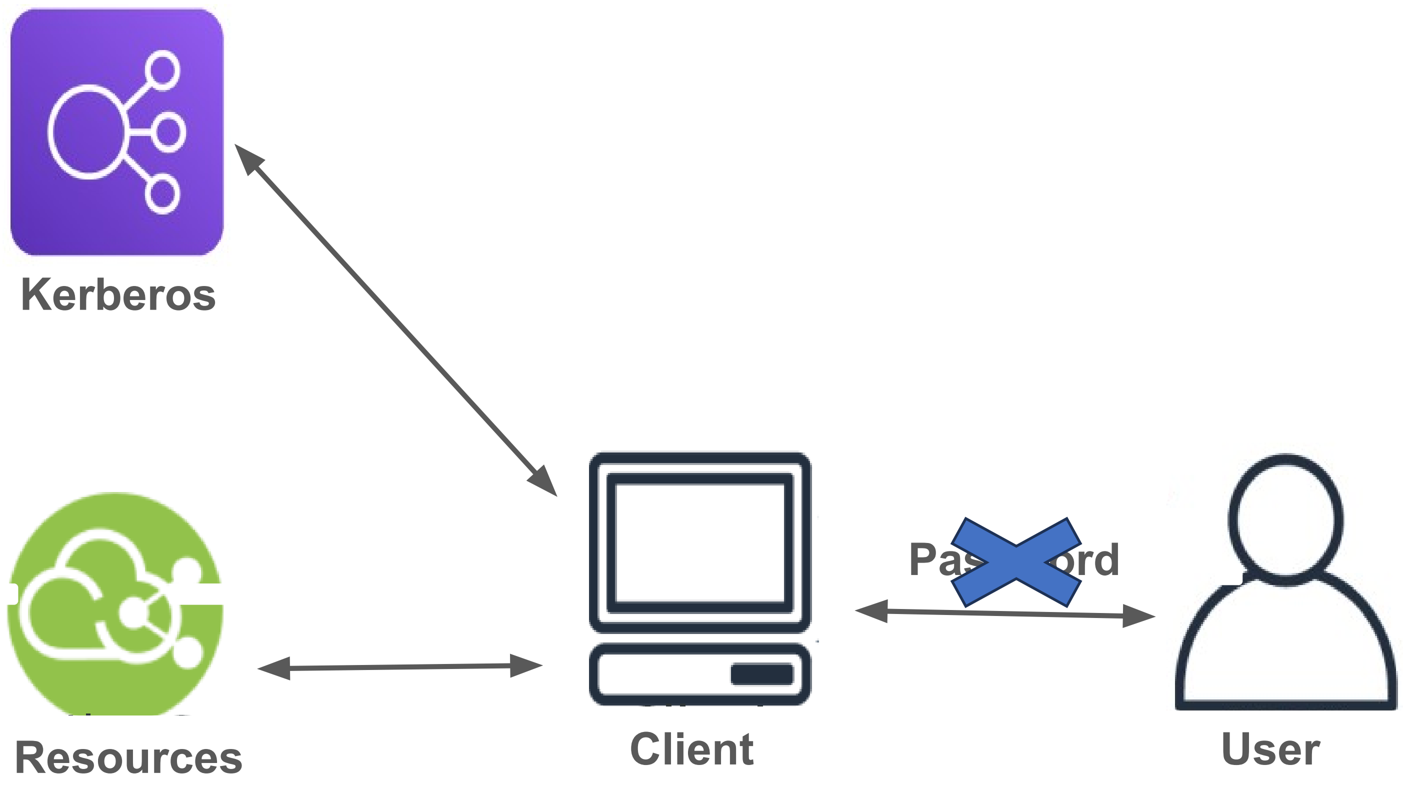
Stealth Server Side

- 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

RSA + PKCS1 V1.5



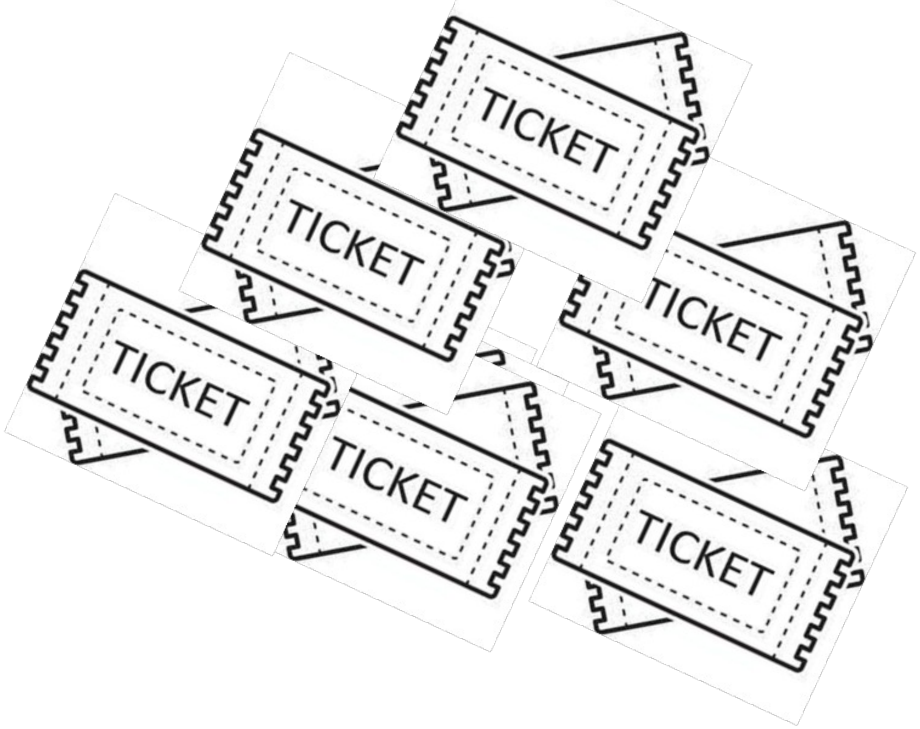
Fast Enough?

- 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

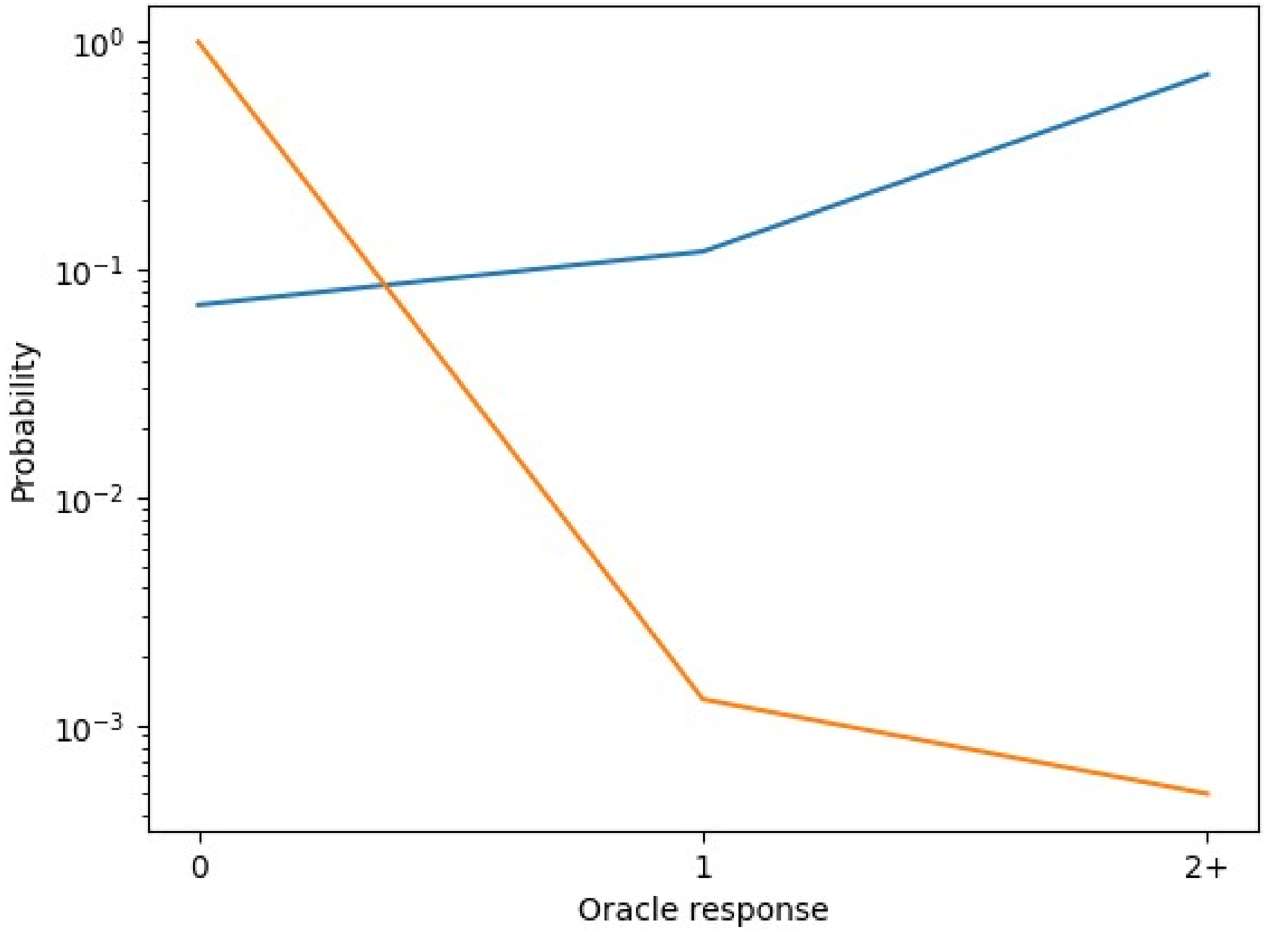
- Range reduction - Additional Multipliers

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

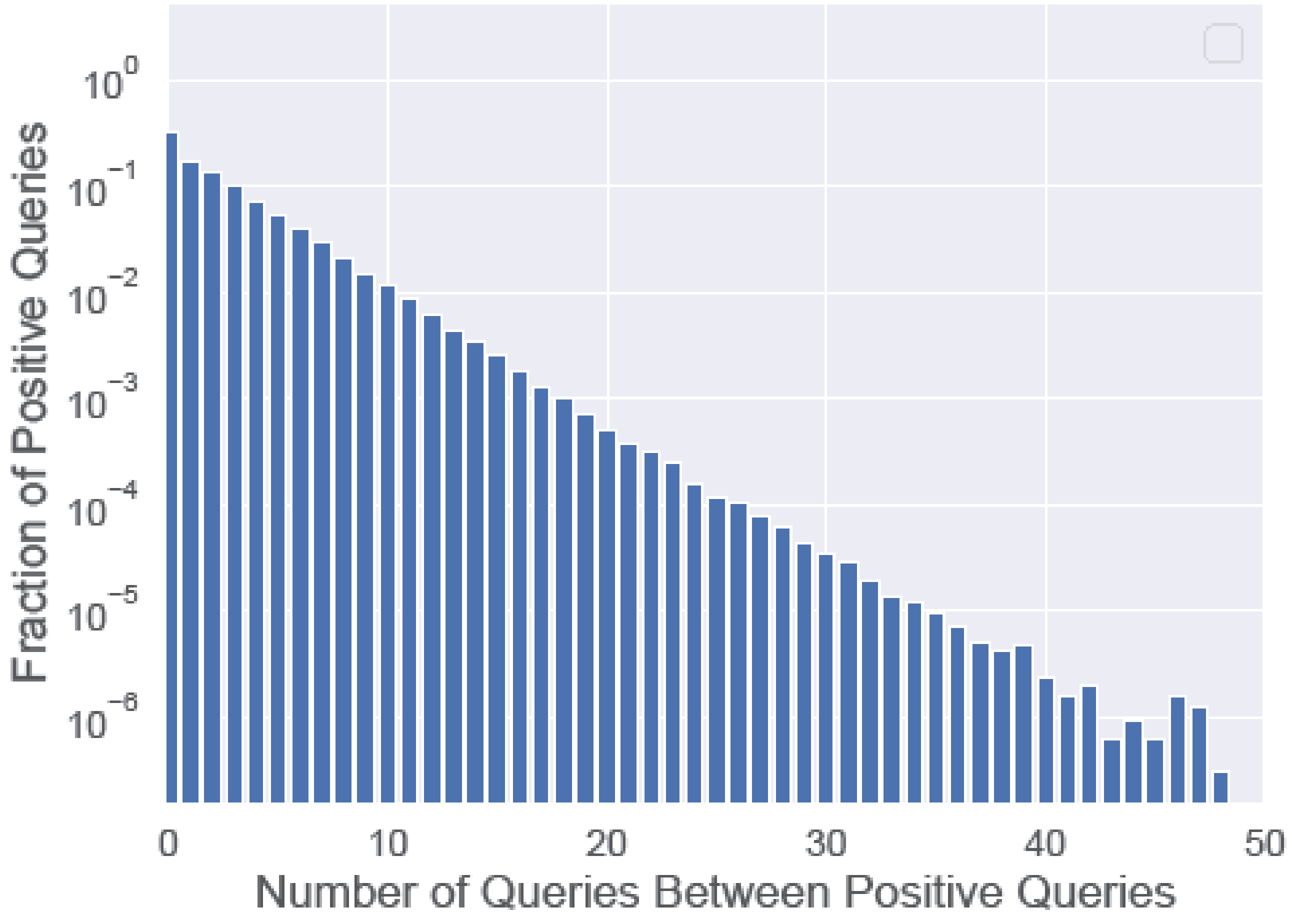


Accelerated Attack- Experiment Results

- 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

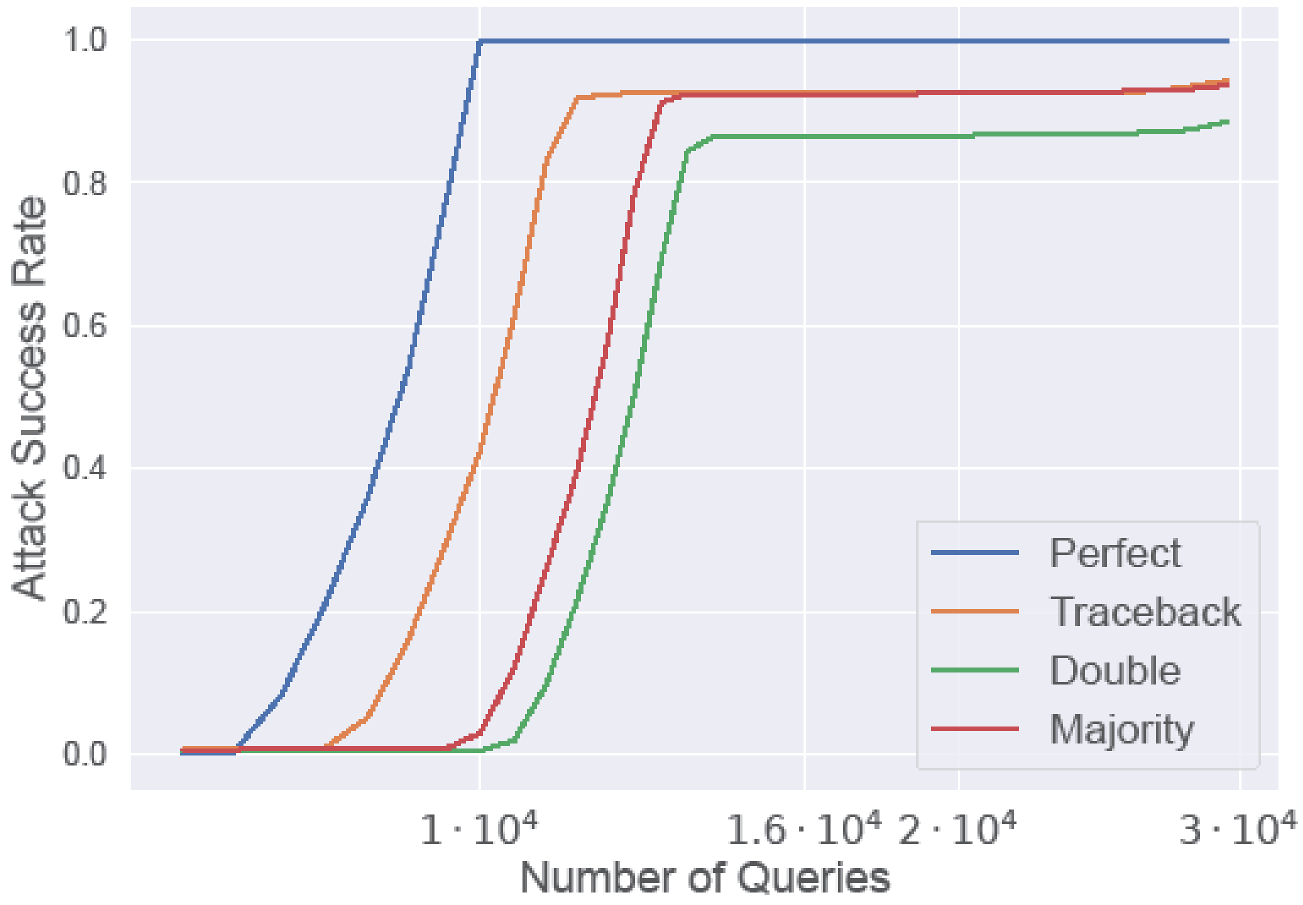
False Positive Detection

- Upper bound for amount of false queries
- Too many false queries in a rov -> 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!

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Git Link