Dynamic Cryptographic Backdoors

Eric Filiol

filiol@esiea.fr

ESIEA - Laval

Operational Cryptology and Virology Lab $(C+V)^O$

CanSecWest 2011 - Vancouver March 9-11th, 2011

Outline

Theoretical Crypto vs Real Crypto

- Secret key size is very often considered as a “key” security feature.
- Blind faith in cryptographic design.
  - “AES-256 inside” marketing syndrom.
  - Necessary but not sufficient condition.
- Religious faith in academic views.
  - “*Give me Eternity, infinite computing power and yobibytes of plain/cipher texts and I can break your crypto*”
  - “*It is strongly secure since it is not broken yet (with respect to the “academic” definition of broken)*”
- But cryptography is a strategic/intelligence matter. Not only an academic playground.
- Efficient techniques are generally seldom published.

Cryptanalysis reality

- What does "to break cryptography" means?
- Use the "armoured door on a paper/cardboard wall" syndrom?
  - The environment (O.S, user) is THE significant dimension.
- Make sure that everyone uses the standards/norms you want to impose (one standard to tie up them all).
- Standardization of mind and cryptographic designs/implementation.
- The aim is it to look beyond appearances and illusions.
- Think in a different way and far from the established/official cryptographic thought.
- To break a system means actually and quickly accessing the plaintext whatever may be the method.

Cryptanalysis reality (2)

- The most simple yet efficient way is use a malware and wiretap the secret key in memory.
  - Windows Jingle attack (Black Hat USA 2008).
  - Do not worry about AVs: they do not detect anything new (just a desktop widget).
- However this simple approach is not always possible
  - E.g. Tempest-protected computers with encrypted network traffic (IpSec, Wifi, sensitive networks [encrypted routers], Tor networks...).
  - Data can be exfiltrated in a single way only: encrypted network traffic which is supposed to be unbreakable.
- It is however to exploit very efficiently the standardization of protocols (IP), cryptographic design, implementations (OS) and of development (crypto API, crypto libraries).

Context and prerequisites

- We present different (not all possible) solutions to break in strongly encrypted/protected networks.
- We rely on the fact that infecting secure networks is (unfortunately) easy.
  - From German Chancelery (2007) to more recent cases (2011)... everywhere.
  - Just send an email with a trojanized attachment (PDF, {Microsoft, Open} Office...).
- We do not recall how to bypass IDS, AV detection. Just use malicious cryptography & mathematics (CanSecWest 2008, H2Hc 2010).
  - Real attacks analyses show that sophisticated malware are always successful.
- We have tested all our PoC against real, strongly protected networks.
- Some codes available upon request. Contact me.

Summary of the talk

Outline

Introduction

- On sensitive networks, the main security objective is to forbid data wiretapping and eavesdropping.
- The most widespread solution is IPSec (or IPSec-like) tunnels.
  - Use of encryption of communication channels.
  - Used in VPN, WiFi...
  - Used in military encrypting IP routers or IP encryptors (e.g. NATO).
  - ...
- Too much confidence in encryption.
  - Why should we use AVs, IDS... (actual observation).
- IPSec-based security is considered as the most efficient one.
- The IPSec standard is very weak and enables attackers to steal data even through an IPSec tunnel.

Introduction (2)

What we are going to demonstrate how:

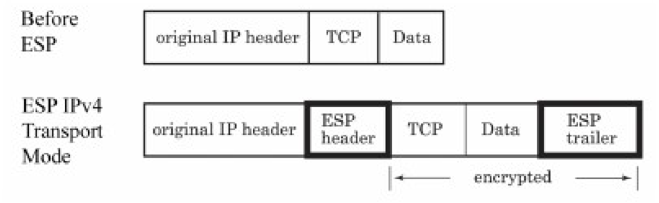
- IPSec-based protocols can be manipulated to make data evade from "secure" computers.
  - Only simple user's permission is required.
- A malware can subvert and bypass IPSec-like protocols.
- Use of a covert channel allowed by the IPSec standards.
- The technique is efficient even on complex traffics (multiplexed traffics, permanent or heavy traffics...).
- Developped in C/Rebol in 2008.

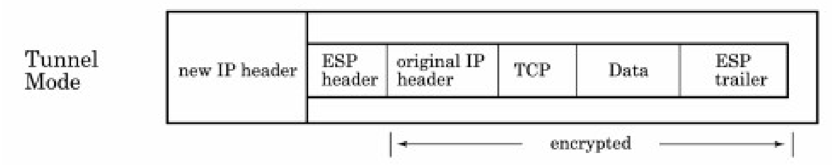
What is IPSec?

IP Security (IPSec) protocol defined by the *Internet Engineering Task Force* (IETF).

- Mostly used to create Private Virtual Network.
- Designed to provide security services for IP.
- Two sub-protocols:
  - `AH` : authentication and integrity.
  - `ESP`: `AH` + data encryption.
- Application-transparent security (*telnet*, *ftp*, *sendmail*...).

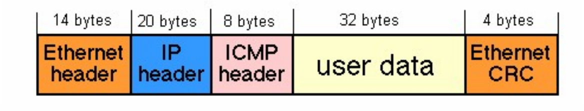
ESP in transport and tunnel mode

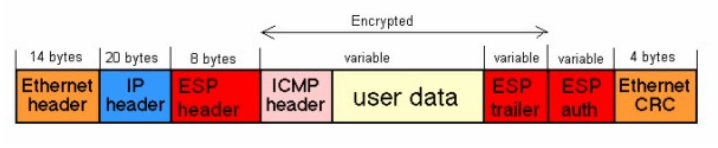




ICMP (Ping) Packet

Our attack essentially considers ICMP (*ping*) packet with ESP encryption in tunnel mode.





Other protocols and covert channels can also be used. But ICMP method is simple and illustrative enough for validation of the general concept.

What is a covert channel

Definition of the US DoD (1985):

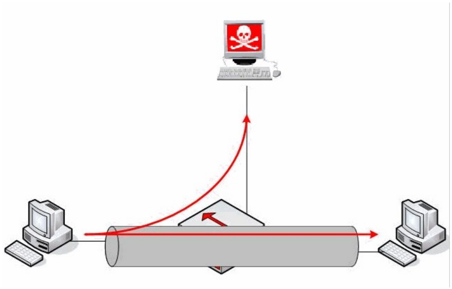
- Communication channel B which borrows part of the bandwidth of an existing communication channel A.
- Enables to transmit information without the knowledge/permission of the legitimate owner of channel A and/or of the data transmitted.
- A few known cases in cryptology:
    - Timing attacks.
    - Power analysis.
    - Side channel attacks...

Previous studies on IPSec covert channels

Only very few (open) studies in this field.

- Packet header manipulation (Ahsan - 2002; Ahsan & Kundur - 2002).
  - The main drawback is packet integrity violation.
- Link between anonymity and covert channels (Moskowitz et al. - 2003)
  - Limited scope due to the lack of control on the IPSec tunnel.
  - Alice and Bob ignores how the network communications are managed.
- Our attack (developped in 2008 with Cridefer & Delaunay).

General Attack Scheme

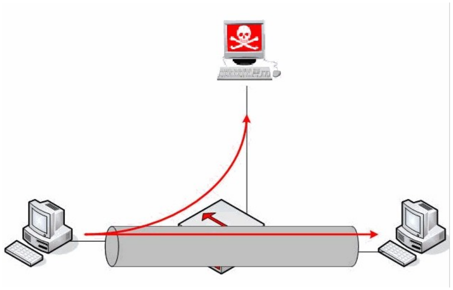


- Alice and Bob communicate through a IPSec tunnel.
- Eve (attacker) wants to eavesdrop confidential data from Alice's computer. She can only observe the encrypted traffic and
  - Extract the IP header added by the IPSec device (e.g. a router in ESP tunnel mode).
  - Get IP packets size.

General Attack Scheme (2)

- Eve deploys a malware which is going to exploit a IPSec covert channel (ICMP-based for exemple).
- The covert channel capacity will decrease with the number of co-emitters.
  - The co-emitters activity will be considered and managed as a transmission noise (error-correcting approach).
- Two-methods are then used by the malware to exploit the covert-channel:
  - The *Ping length method*.
  - The error-correcting codes-based optimized *Ping length method*.
- Very efficient method to make file/emails evade from Alice's computer.

The *Ping length method*



- One-to-one correspondance between data characters to evade and ICMP packet sizes.
- Eve wiretaps the encrypted traffic and extracts the packet size to decode the data.
- Coding/decoding techniques must be powerful enough to cancel the noise.
- Two-part malware: *AlphaPing* (Alice) and *AlphaServer* (Eve).

AlphaPing Side

- Collects the data to evade (binary files are base64-encoded).
- Each character is repeated five times (5-repetition code).
- Use of dedicated traffic tags:
    - *Begin* tag.
    - *Stop* tag.
- To optimally manage the IPSec protocol (8-byte encryption), ping packet sizes must differ from at least eight units.
- Written in Rebol (*Relative Expression-Based Object Language*). A powerful network-oriented language with lightweight interpreter.
- The size of *AlphaPing* (in Rebol) is 960 bytes.

AlphaPing Side (2): character encoding

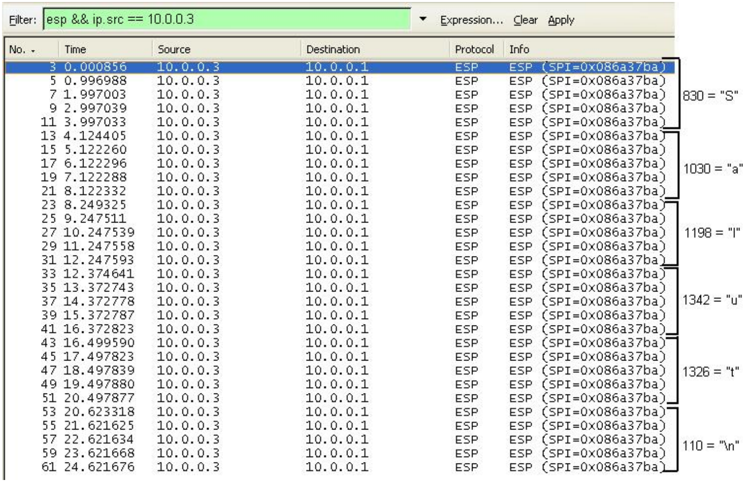
- Simple encoding ping packet size $\leftrightarrow$ character value for text files.
- Binary files are first base64-encoded.

ping packet size $\leftrightarrow$ character value mapping

```
switch (length) {
case 102: return '\t';
case 110: return '\n';
...
case 598: return 'A';
case 614: return 'B';
case 622: return 'C';
...
```

AlphaPing Side (3)

Emission of the character string “Salut” (5-repetition code).



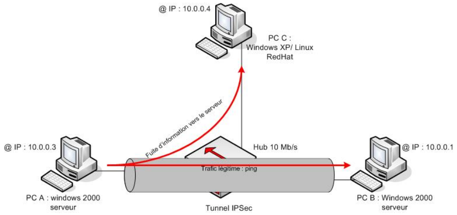
```
Filter: esp && ip.src == 10.0.0.3          Expression...  Clear  Apply

No.   Time        Source      Destination  Protocol  Info
  3 0.000856     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
  5 0.996988     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
  7 1.997003     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)   830 = "S"
  9 2.997039     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 11 3.997033     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 13 4.124405     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 15 5.122260     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 17 6.122296     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)   1030 = "a"
 19 7.122288     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 21 8.122332     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 23 8.249325     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 25 9.247511     10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 27 10.247539    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)   1198 = "l"
 29 11.247558    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 31 12.247593    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 33 12.374641    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 35 13.372743    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 37 14.372778    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)   1342 = "u"
 39 15.372787    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 41 16.372823    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 43 16.499590    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 45 17.497823    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 47 18.497839    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)   1326 = "t"
 49 19.497880    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 51 20.497877    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 53 20.623318    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 55 21.621625    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 57 22.621634    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)   110 = "\n"
 59 23.621668    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
 61 24.621676    10.0.0.3    10.0.0.1     ESP       ESP (SPI=0x086a37ba)
```

AlphaServer Side

On Eve's side, she.

- Passively observes the packet flow and extracts suitable packets by using 5-repetition decoding techniques (ML decoding).
- Reverses the packet size/character mapping.
- Base64 decodes the resulting message.
- 5-repetition codes are powerful enough in most cases but noise reduction can be optimized by using suitable coding/decoding techniques (error-correcting codes-based optimized *Ping* length method; technical details available upon request).

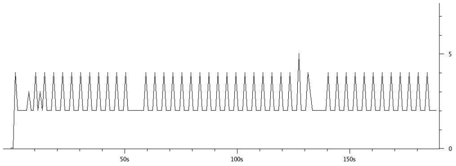
Test Platform


@ IP : 10.0.0.4
PC C : Windows XP/ Linux RedHat
Fuite d'information vers le serveur
@ IP : 10.0.0.3
Hub 10 Mb/s
@ IP : 10.0.0.1
Trafic légitime : ping
PC A : windows 2000 serveur
Tunnel IPSec
PC B : Windows 2000 serveur

- Packet analyzer (*Wireshark*).
- Tunnel activity monitor (*ipsecmon*).
- Automated traffic generator to simulate different traffic load.

Experimental Results: Normal Traffic Load

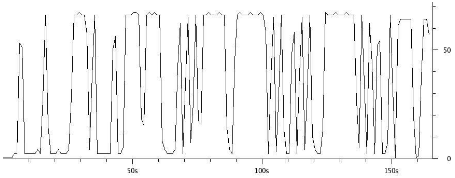
The message “Salut comment ca va aujourd’hui ?” is emitted by the malware.



- Wireshark analysis: traffic load with respect to time.
- No residual error.
- Total transmission time = 145 seconds.
- Should be easy to detect by good IDS (no TRANSEC).

Experimental Results: continuous random load (1Kb/s)

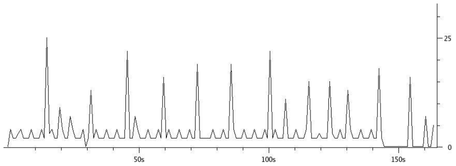
The message “Salut comment ca va aujourd’hui ?” is emitted by the malware.



- Many errors (without decoding techniques).
- Total transmission time = 165 seconds.
- Can no longer be detected by IDS (traffic load hides malicious emission).
- Most usual cases (multi-user network).

Experimental Results: 4 Kb/s burst with random phase

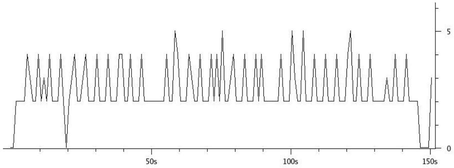
The message “Salut comment ca va aujourd’hui ?” is emitted by the malware.



- A few errors (without decoding techniques).
- Total transmission time = 145 seconds.
- Can eventually be detected by IDS (weak TRANSEC).

Experimental Results: traffic with Random Burst

The message “Salut comment ca va aujourd’hui ?” is emitted by the malware.



- Two residual errors (“Salut commenB ca Aa aujourd’hui ?”) without error-correction.
- No transmission time increase.
- Difficult to detect with IDS.

Optimizations

- How to bypass IDS detection?
- How to optimally correct residual decoding errors?
- The *AlphaPing* part is going to use heavily loaded traffics.
    - However, we have observed that on most real networks the traffic load is high enough to hide our malicious communication.
- To decode without residual errors, new coding/decoding schemes must be used.
- Use of more sophisticated data synchronisation/tagging techniques based on combinatorial patterns (needs more maths you would accept to tolerate/accept here -:))
- Data are encoded under their hex value.

Optimizations: Efficient data encoding

Efficient one-to-one character/size mapping:

Character	0	1	2	3	4	5	6	7
Packet length	160	176	192	208	224	240	256	272

Character	8	9	A	B	C	D	E	F
Packet length	284	300	316	640	656	672	688	704

- Efficient at bypassing IPSec fragmentation effect. Packet size values are limited to a reduced interval ($[160, 704]$).
- Use of n-repetition codes (among the most powerful error-correcting codes).

Optimizations (2): n repetition codes

Suppose that in most traffics (sufficient as first approximation), packet sizes are uniformly distributed (note that the malware can perform a prior statistical analysis of the output traffic to recover the actual probability law; as Eve can as well).

Let us denote by p_i the probability of occurrence of a packet of size i (under the uniform law hypothesis $p_i = \frac{1}{1514}$). In a "window" of p packets ($n < p$),

- In normal conditions (e.g. without the malware) a (non necessary contiguous) pattern of n times the packet size s occurs in average $\binom{p}{n}.p_i^n$.
- According to the traffic load (which has an impact on the window size p) then choose the value n such that this probability is negligible.
- Experiments have shown that for most traffics $n \in \{5, 7, 9, 11\}$ the residual decoding error probability tends towards 0.

Comments (1)

- Other protocols than ICMP can be also used (DNS requests, HTTP requests, TTL, hop limit...).
- Detection with IDS (e.g. Snort) is impossible (untractable to monitor all possible protocols/streams/methods especially for heavily loaded traffics).
- More sophisticated combinatorial coding/decoding techniques are possible to
  - To manage heavily loaded traffic with a large number of co-emitters.
  - Reduce the bandwidth consumption of the covert channel.
  - Reduce the network signature.
- Malware network-adaptative behaviours (to the traffic load for exemple).

Comments (2)

- Security provided by IPSec is illusory in most cases.
- Powerful methods for passive eavesdropping in any kind of traffic.

To protect against the *Ping length method*, the best method is:

- Armoured version of IPSec protocol with systematic padding to have the maximal (unique) packet size available.
    - Only a few devices are using systematic padding (*NetAsq*, *Harkoon*, IP encryptors...).

Outline

Introduction

- How to bypass security enforced in very secure encrypted protocols (e.g. IP encrypting routers with systematic padding)?
- The first solution is to exploit the fact that many encryption algorithms rely on the operating system primitives to generate secret keys (e.g. Microsoft cryptographic API).
- The second solution is to modify the cryptographic algorithm on-the-fly in memory:
  - Its mode of operation and/or its mathematical design.
- The algorithm is not modified on the hard disk (no static forensics evidence).
- The trapdoor has a limited period of time and can be replayed more than once.
- In both cases, the encryption has been weakened in such a way that
  - the attacker has just to intercept the ciphertext and perform the cryptanalysis.

OS Level Dynamic Trapdoors

- Here we considered strong cryptosystems (AES, TrueCrypt, GPG/PGP...).
- However the security at the operating level is not perfect.
- What is it possible to do with a simple malware?
- What about computers with no network connection or whenever key wiretapping is no longer possible?
- The “static (mathematical) security” remains unquestioned!
- Just create dynamically periods of time during which the encryption system is weak.
- Techniques developped by Baboon and myself.

Program Interaction Control

- Here we exploit the fact that very often, the message key K_m is built from data provided by external programs.
  - Message counter, message key, session key...
  - Initialization vectors for block ciphers.
  - Integer nonces.
- Most of the time the resources involved are in the Windows API.
  - They provide random data required by the encryption application to generate message keys and IVs
- You then just have to hook the API function involved.
- Same approach for other equivalent resources (key infrastructure, network-based key management...).

Hooking the CryptGenRandom function

- Drawn from a real case (see further).
- A malicious DLL is injected in some (suitable) processes. This DLL hooks the CryptGenRandom function (included in Microsoft's Cryptographic Application Programming Interface).

CryptGenRandom function

```
BOOL WINAPI CryptGenRandom(
__in HCRYPTPROV hProv,
__in DWORD dwLen,
__inout BYTE *pbBuffer
);
```

- A timing function checks whether we are in the time window given as parameter $sTime(12, 00, 14, 00)[...]$. will hook the CryptGenRandom function between noon and 2pm only.

Hooking the `CryptGenRandom` function (2)

- The integer (random data) returned by `CryptGenRandom` is modified by the function `HookedCryptGenRandom`.
  - They provide random data required by the encryption application to generate message keys and IVs
- You then just have to hook the API function involved.
- Same approach for other equivalent resources (key infrastructure, network-based key management...).
- On Bob's side, the ciphertext can still be deciphered.

Hooking the CryptGenRandom function (3)

Generate fixed message key $0x1212121212121212$

HookedCryptGenRandom function

```
BOOL WINAPI HookedCryptGenRandom(HCRYPTPROV hProv, DWORD
dwLen, BYTE *pbBuffer)
{
static BOOL send12 = 0; BOOL isOK; DWORD i;
send12 ^= 1;
isOK = HookFreeCryptGenRandom(hProv, dwLen, pbBuffer);
if((send12) && (isOK))
for(i = 0; i < dwLen; i++) pbBuffer[i] = 0x12;
return isOK;
}
```

How to Exploit this

- For stream ciphers and block ciphers in stream cipher modes (CFB, OFB, CTR), making the message key or IV constant produces "*Parallel ciphertexts*" during a limited period of time.
  - Easy to detect and break (PacSec 2009 - Black Hat Europe 2010) (polynomial time).
  - Use the cryptanalysis library Mediggo `http://code.google.com/p/mediggo/`.
- Main drawback: it does not apply to ECB, CBC modes.
- But (some) cryptographic APIs make things easy if you know where to look.
- Most of the cryptographic APIs have been "inspired" by the *NIST AES Cryptographic API Profile*.
- This standardization of developpers' mind enables powerful attacks for a number of implementations.

Modify the cryptographic algorithm

You can also patch the algorithm on-the-fly to modify

- Its operation mode
  - Turn CBC/ECB modes into OFB/CFB/CTR mode (sometimes requires a limited amount of modifications).
  - Many implementations (more than expected) concerned.
- Its internal (mathematical) design
  - Selectively modify one or more Boolean functions
  - Change all or part of the S-Boxes.
- On Bob's side, of course the ciphertext is no longer decipherable, unless Alice AND Bob have been infected (targeted attack).
- If the window of time is very limited, this can be seen as an internal error or wrong password used. Alice and Bob will just exchange the message one more time.

Operation mode modification

General scheme (inspired from real cases)

```
int cipherInit(cipherInstance* cipher, BYTE mode, char* IV) {
switch (mode) {
...
case MODE_CFB1:
...
}
int blockEncrypt(cipherInstance* cipher, keyInstance* key, BYTE*
input, int inputLen, BYTE* outBuffer) {
....
switch (cipher->mode) {
...
case MODE_CFB1: ...
}}
```

Only a few modifications are required to switch to CFB1 mode (set argument *BYTE mode* to 3)..

Modify the internal design

- The idea here consists in scanning for active encryption system in memory and modifying their mathematical design on-the-fly only.
- Volatile modification which does not affect the application on the disk.
- Our Implementation to attack AES
  - `scanKernelModules` function to look for AES' sboxes signature.
  - `patchModule` function to modify (weaken)/change the Sboxes.
  - `writeModule` function to bypass write-protection of memory page.
- You can do many other things
  - ... no limit but your imagination!

PoC

- Use of k-ary malware. Very powerful computer malware (Journal in Computer Virology, 3(2), 2007 - Hack.lu 2009).
- A k-ary malware ($k = 4$) has been designed (parallel mode, B class).
  - Detection of k-ary malware is at least NP-complete.
- First part just turns CBC into CFB.
- Second part hooks the `CryptGenRandom` function.
- The two other parts provide anti-antiviral protection.
- The malware operates during a limited period of time (dynamic trapdoor).

Outline

Conclusion and Future Works

- Cryptographic security more than ever relies more on the algorithm environment than on the algorithm itself.
- The power of standards and norms must not be underestimated.
- Check (software/hardware) implementation carefully.
- What the solution?
  - Hardware-based hypervised OS could prevent on-the-fly algorithm patching techniques (current development for the French industry).
  - Use an additional IP encryptor with packet padding.
- To be continued...

Thanks and credits

Thanks to all those who have contributed to this study.

- Guillaume Delaunay.
- Cridefer.
- Baboon.

Many thanks for your attention.

Questions and answers!