# NFSv4 (Network File System) : Security audit of the GNU/Linux implementation

*Jonathan LYARD*

*Intern at Bull (Grenoble – France)*

Aug. 24, 2006

# Abstract

- **Working context : NFSv4 team**

- **Security audit**
  - Functional
  - Theoretical audit (cryptographic study)
  - Source code audit in the Linux kernel
  - Dynamic audit (fuzzing and fault injection techniques)


© Bull, 2006

# The NFSv4 protocol

- A new version of the Network File System

- Caching and delegation
  - Client-side caching
  - Open delegation
  - Client callbacks
- Locking
- Attributes
  - Mandatory
  - Recommended
  - Named (ACL)
- Security model
- Negotiation of security flavors
- Migration and replication
- Minor versioning
- Modifications for use on WAN/Internet


© Bull, 2006

# Bull's contribution

- **Team managed by Tony Reix**

- **Aurélien Charbon**
  - Currently : Development of IPv6 client and server
  - Before : Test of the ACL implementation

- **Aimé Le Rouzic**
  - Test of kernel patched with latest CITI's NFSv4 support
    - Functional
    - Stress
    - Performances
    - WAN
    - Interoperability


© Bull, 2006
Bull

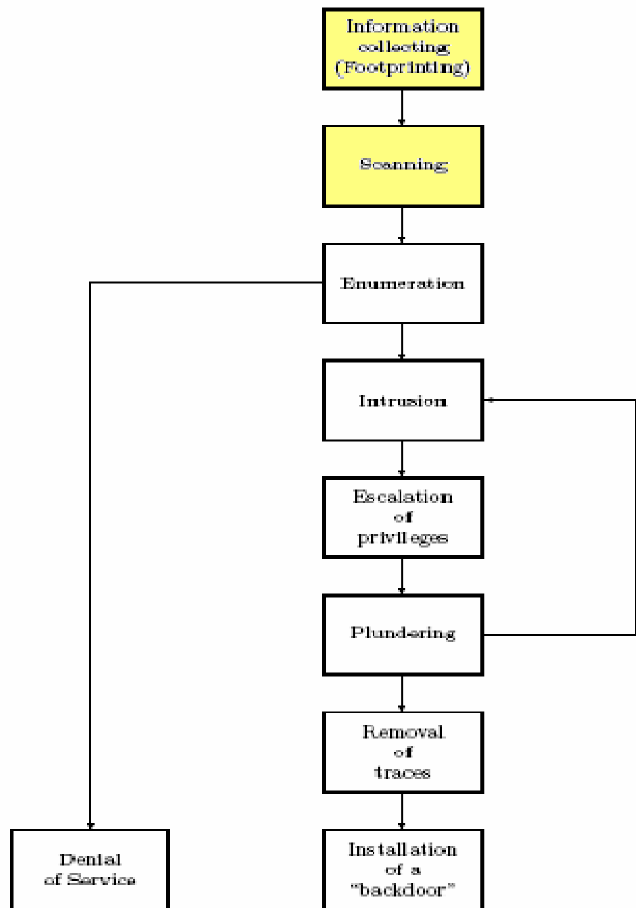
# Security audit of NFSv4

- **Needs for a security audit**
  - A widely used protocol
  - WAN/Internet context
  - State of the security analysis
- **Goals**

© Bull, 2006

Bull

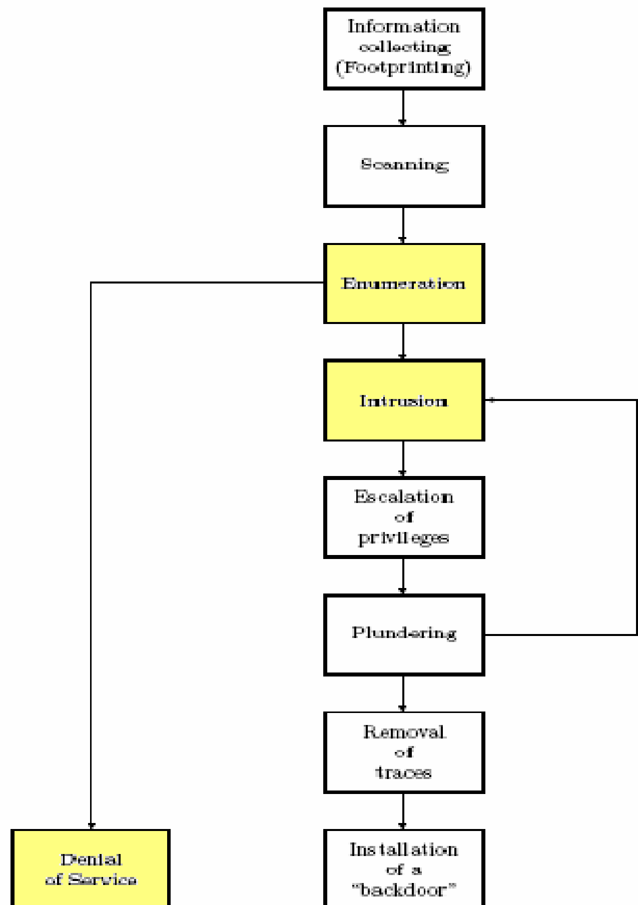
# Data gathering




- **Detecting and gathering information about NFSv4 server is easy :**
  - rpcinfo on TCP 111
  - Usage of a well defined port (2049) that can be scanned (i.e nmap)
  - NULL NFSv4 requests
  - Minor version negotiation


© Bull, 2006

# Adversary model

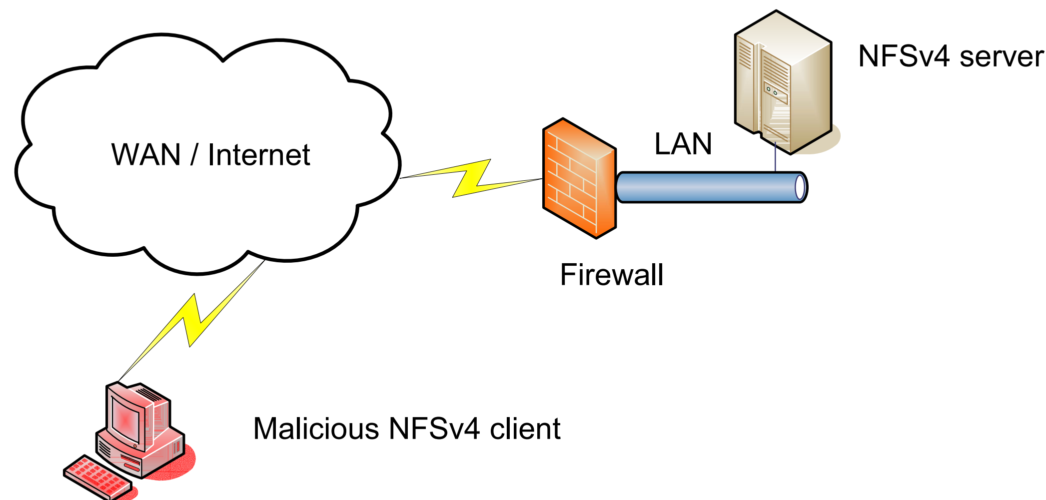



- **Several attack scenarios :**
  - Modified NFSv4 client
  - Attack against NFSv4 request/response
  - Taking advantage of a NFSv4 server
  - Side-channel attacks


© Bull, 2006

# Adversary model (1)

- **Attack from a modified NFSv4 client (Risk : High)**

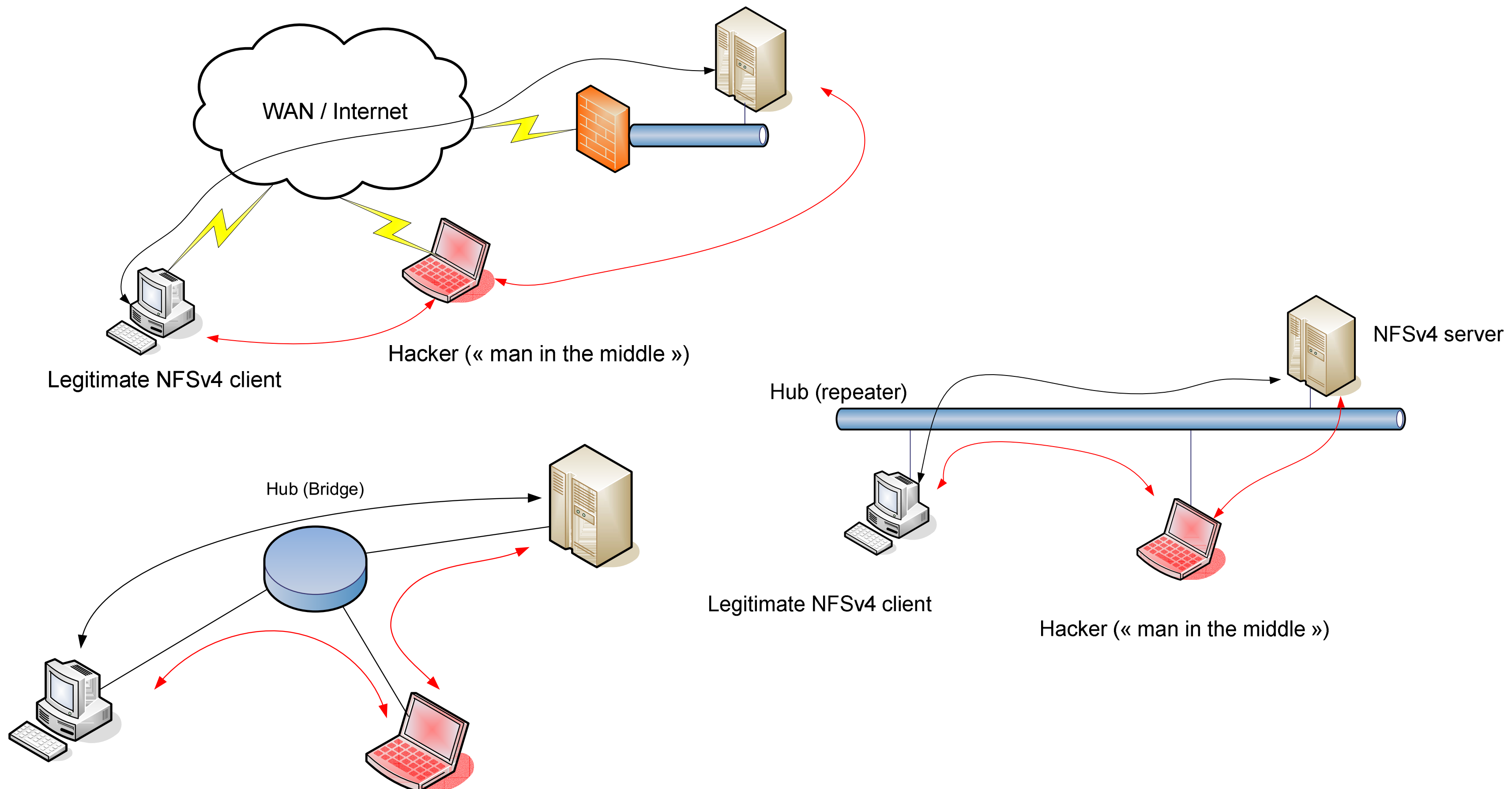



- **Counter measures :**
  - **Security protocol : RPCSEC_GSS or external (IPSec, SSH tunneling)**
  - **Source code audit from NFSv4 community**
  - **patching (minor versions)**


© Bull, 2006

# Adversary model (2)

- **Attack against NFSv4 request/response (Risk : High)**





© Bull, 2006

# Adversary model (2)

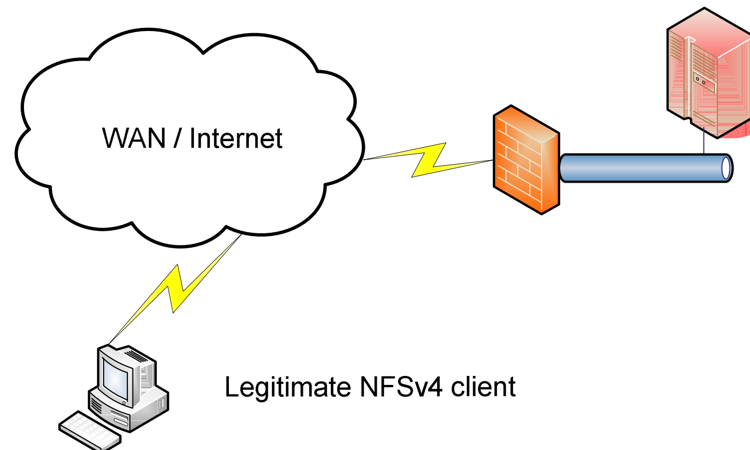
- **Hacking methods :**
  - **LAN :**
    - sniffers in promiscuous mode (Ethereal, TCPDump…) inside the collision domain (Repeater)
    - ARP spoofing, ARP Cache Poisoning, ICMP Redirect outside the collision domain (Bridge)

    *Recommended tool : Ettercap (http://ettercap.sourceforge.net)*
  - **WAN**
    - DNS Spoofing
    - Attack routing protocols (hard if OSPF and BGP use authentication)
- **Counter measures**
  - Authentication and encryption of NFSv4 traffic
  - Control who is connected on the LAN (wireless networks, laptops on Ethernet networks…)
  - Authenticate and protect integrity of routing advertisements


© Bull, 2006
Bull

# Adversary model (3)

- **Taking advantage of a NFSv4 server** (Risk : Low)
  - The attacker already owns a user account
  - The attacker sets up a malicious NFSv4 server (NFSv4 ~ FTP)

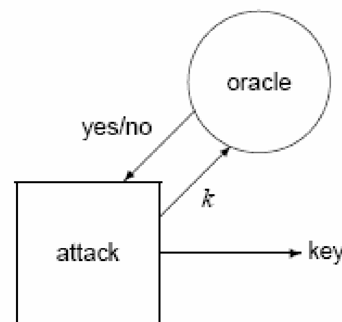



- **Counter measures :**
  - Authentication of NFSv4 server (to avoid being redirected on a malicious one)
    - Public Key Infrastructure (PKI) : currently NFSv4 supports SPKM3 to authenticate both the client and the server thanks to certificates
  - Audit of NFSv4 client's source code by the community


© Bull, 2006

# Adversary model (4)

- **Side channel attacks** **(Very Low)**
  - The attacker can probe CPU utilization of svcgssd and gssd
  - The attacker spies NFSv4 exchanges and probes the response time
  - The attacker can send arbitrary NFSv4 requests and can get the response in return.
    - NFSv4 server can be used as a “stop test oracle” for brute force attack against the session key (krb5)



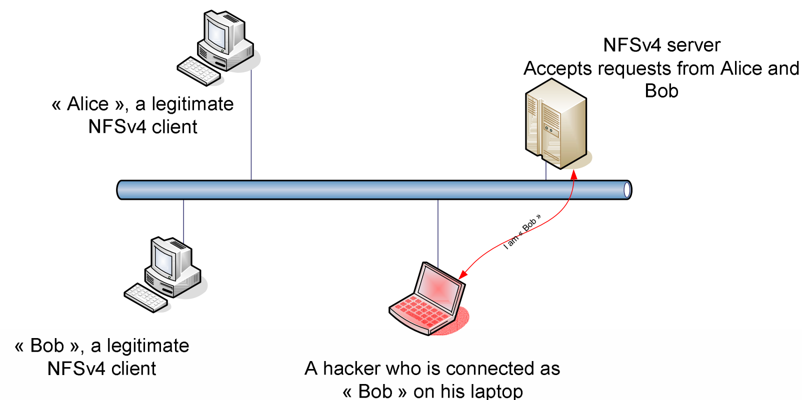


© Bull, 2006

# Unix authentication in the RPC protocol

- **UNIX (SYS) authentication based on UID/GID.**

- **Can easily be defeated :**
  - A hacker who owns a legitimate account on a machine in the network can easily switch to another UID/GID :
    - Attack against password hashes
    - Vulnerabilities in local programs to spawn a root shell
  - Usage of RPC generators to forge RPC requests including NFSv4 request
    - Experimented with SPIKE
  - Physical intrusion (wireless networks, laptops…)
    - the hacker can impersonate any user





© Bull, 2006

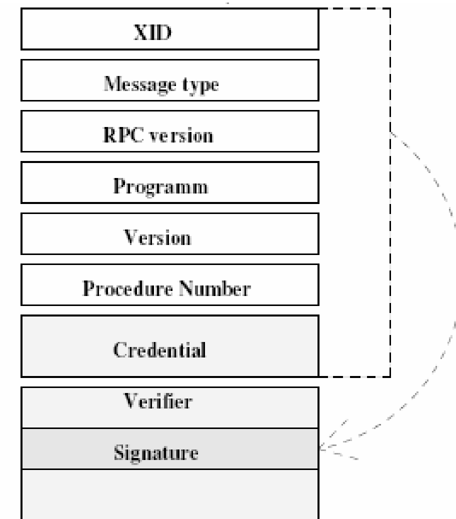
# 3 security flavors in RPCSEC_GSS

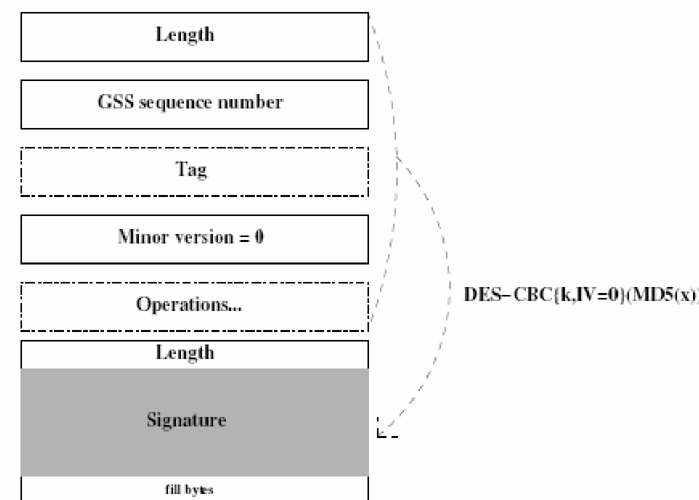
- **RPCSEC_GSS** is a generic protocol which uses either :
  - **Kerberos 5** : authentication relies on a third-party, called a KDC, which delivers the session key
    - So far, the protocol is considered secure
    - Problems to extend realms to WAN networks (authority) and to share authentication between different realms → can't be used on public networks (WAN, Internet)
  - **SPKM3** : usage of certificates to authenticate both sides
    - Certificates should be exchanged in a secure way !
  - **LIPKEY** : on top of SPKM3, to add password authentication of the client


© Bull, 2006

# 3 security modes in RPCSEC_GSS

- **Authentication** : a MAC (Message Authentication Code) is computed on the RPC header
  - → *Protects from impersonation*
- **Integrity** : a first MAC is computed on the RPC header and another is computed on NFSv4 data
  - → *Protects from data corruption*
- **Privacy** : the 2 MACs + encryption of NFSv4 data
  - → *Protects from sniffing*
    - → *(remote home directories with .ssh)*



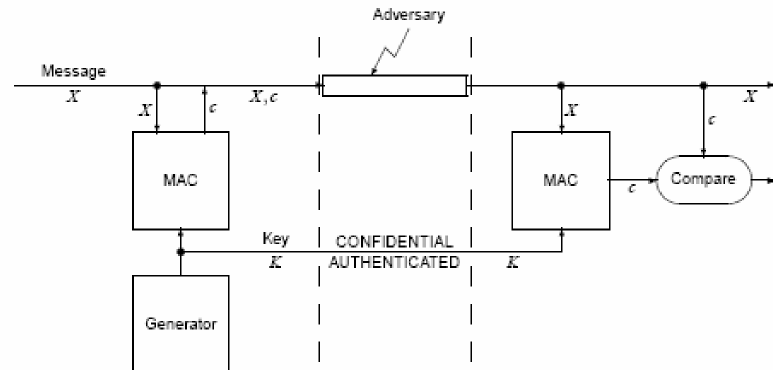



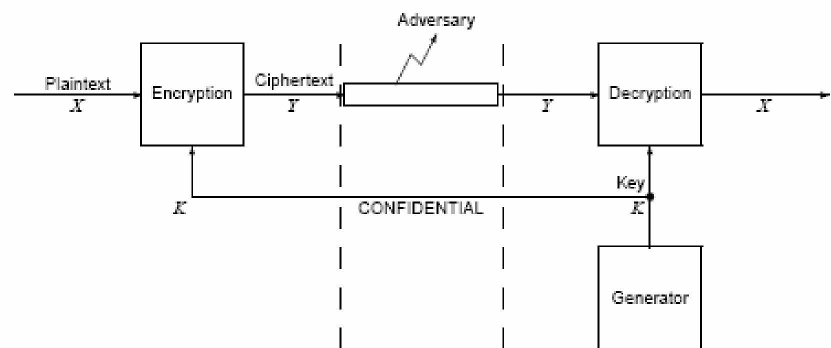

 © Bull, 2006

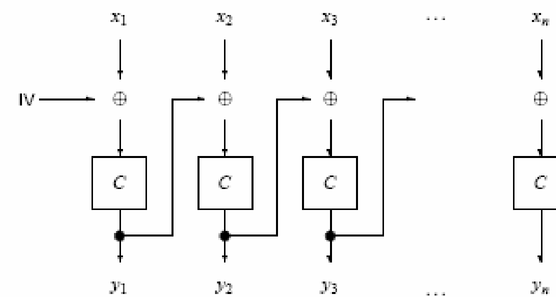
# Threats against RPCSEC_GSS

- **Threats against MACs :**
  - Forge a valid MAC for an arbitrary message X
  - Find a couple (X,c) with some given properties on M
  - Replay attacks
  - Swap $(X_1,c_1)$ and $(X_2,c_2)$

- **Threats against encryption :**
  - Brute force
  - Information leakage
    - On the secret key
    - On the plain text
    - ⇒ DES-CBC($x_1$ | $x_2$ | … $x_n$) = $y_1$ | $y_2$ … | $y_n$
    - ⇒ DES-CBC($x_1$ | $x_2$‘ | … $x_n$‘) = $y_1$ | $y_2$‘… | $y_n$‘




 © Bull, 2006

# Why is RPCSEC_GSS not secure enough ?

- **MD5 hashing can be defeated in less than $2^{64}$ due to weaknesses in the algorithm**
  - SHA-1 also has some weaknesses but is much better
- **The attacker can play with the hash function : he inputs valid NFSv4 requests to hopefully find collisions**
  - We proved that collisions exist on some plaintext spaces and that it is feasible to find them
  - ISO/IEC 9797 standard should be used to build a MAC from a block cipher
- **DES encryption with a 56-bit keys is not secure**
  - We proved that an encrypted NFSv4 request is decrypted in a single plaintext NFSv4 request (with high chances)
  - DES has some weaknesses (weak keys, semi-weak keys…).
  - DES can be cracked in $2^{43}$ with linear cryptanalysis (instead of $2^{56}$)
  - AES or 3-DES with a 128-bit key should better be used

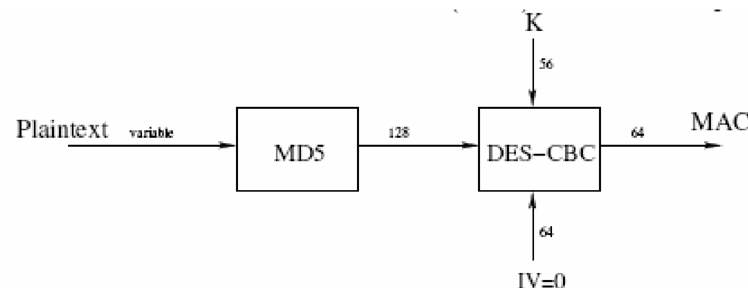

© Bull, 2006
Bull

# Weaknesses in the MAC

**Input:** a cryptographic hash function $h$ onto a domain of size $N$

**Output:** a pair $(x, x')$ such that $x \neq x'$ and $h(x) = h(x')$

1: **for** $\theta\sqrt{N}$ many different $x$ **do**
2: compute $y = h(x)$
3: **if** there is a $(y, x')$ pair in the hash table **then**
4: yield $(x, x')$ and stop
5: **end if**
6: add $(y, x)$ in the hash table
7: **end for**
8: search failed




Complexity of a collision search on the hash function < $\sqrt{2^{128}} = 2^{64}$

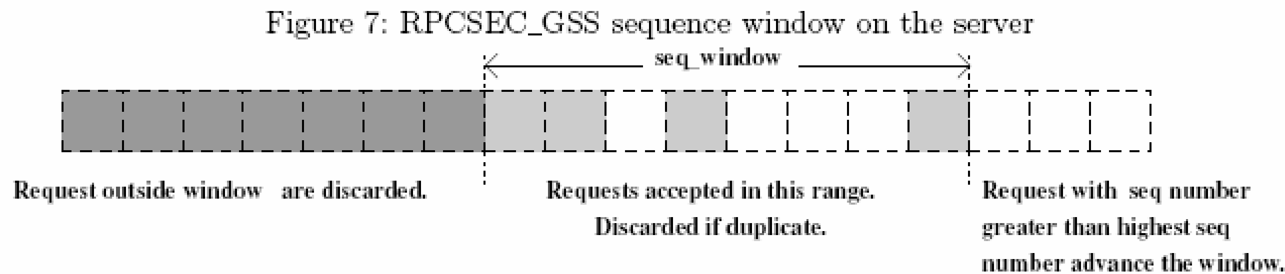
Collisions on the hash function are key-independent

Output space cardinality is $2^{64}$ : an attacker spying $2^{32}$ requests has one chance out of two to find a collision.


© Bull, 2006

# Other attacks

- DoS (Deny of Service) using sequence number




- Message stealing attacks :
  - Steal request
  - Steal response
    - the duplicate request cache will hopefully detect the retransmission, else it can be critical for non-idempotent requests


© Bull, 2006

# Source code audit

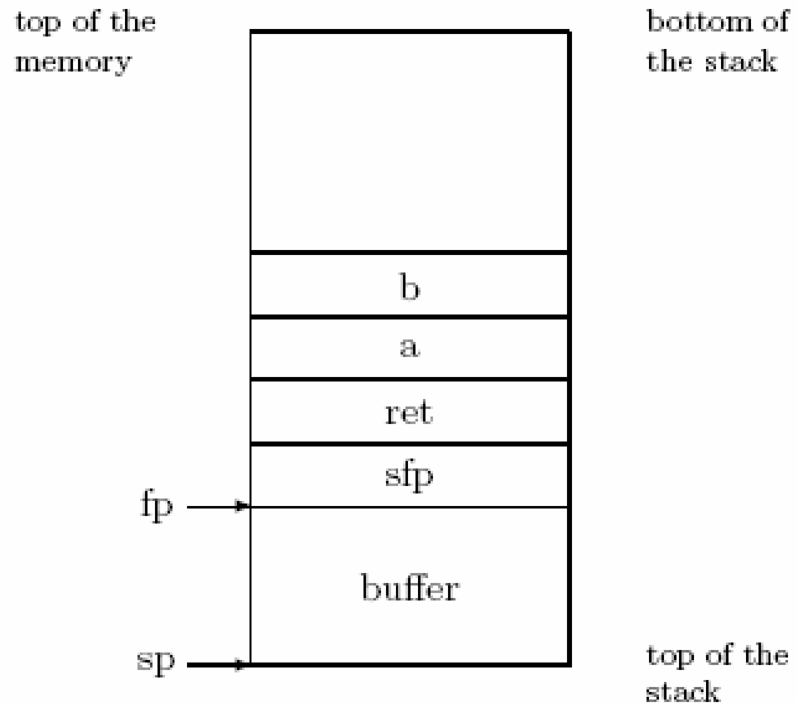
- Why does NFSv4 source code need to be audited ?
- We listed the C vulnerabilities we had to look for
  - Overflows, format string bugs, off-by-one, signed comparison, integer overflows, out-of-scope memory usage, use after free…
- Automated source code analysis. Flawfinder was chosen :
  - Few false negatives
  - Lot of false positives (drawback)

→ Each potential vulnerabilities must be investigated

- Audit area :
  - Client / kernel code (fs/nfs) : 25,000 lines
  - Server / kernel code (fs/nfsd) : 18,000 lines
  - User code and lots of librairies


© Bull, 2006

# Buffer overflows




```
void f(int a, int b) {
   char buffer[10];
   strcpy(buffer, ‘‘hello world’’.);
}

void main() {
    f(1,2);
}
```


© Bull, 2006

# Exploitation of buffer overflows

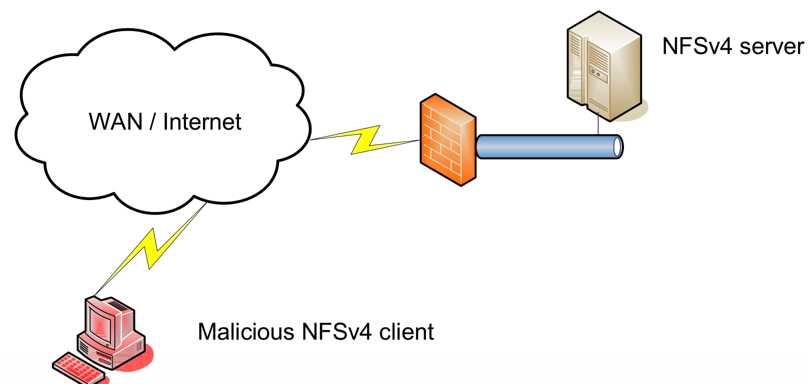
- Write the code of the exploit in assembler. Basically, it will just spawn a shell : it should call execve with “/bin/sh” as parameter.
- Assemble the code to get opcodes and eliminate all null bytes : the shellcode is now injectable
- Convert the injectable shellcode into a string if necessary (i.e. HTTP) *Not necessary for NFSv4.*

- There are additional difficulties, for instance:
  - Guess the address where our shellcode will be located on the remote machine’s stack (i.e. NOP method)
  - GCC versions > 3 use stack protections by default so that it is harder to exploit buffer overflows (but still feasible…)


© Bull, 2006
Bull

# Attack NFSv4 server with a modified client

- What is a dynamic audit ?
- Draft of a modified client using SPIKE framework
- Difficulties and future work
  - NFSv4 is stateful (i.e fid)





© Bull, 2006

# Conclusion

- Security of NFSv4 in itself seems to be fine
- NFSv4's security protocol is not secure anymore (specification was done in 1997!)
  - Developers will provide a new version of RPCSEC_GSS soon
- The dynamic audit has to be continued…
- There are still lot of source code (client, librairies…) to be audited.

- Attacking NFSv4 is harder than attacking Web-based software for instance. There are little chances that hackers will chose this entry point to attack a machine. But still…


© Bull, 2006
Bull