An (In)security Overview on Analysis of Client-Server Software Applications

Author: Giuseppe 'Evilcry' Bonfa'
E-Mail: evilcry {AT} gmail {DOT} com
Website: http://evilcry.netsons.org
Blog: http://evilcodecave.wordpress.com
PublicKey: End of Doc

Introduction

The principal objective of this paper is to give a good detailed panoramic view of the Security aspects involved in **Client-Server based Applications**. The panoramics will be seen from the point of view of a Reverse Engineer that should be aware of the **Security Problems** that are directly releated to the Client-Server Software Structure**.** We will try to cover the full spectrum of C-S software vulnerabilities that may be categorized as follows:

- **Intrusion Risks**
- **Sensitive Data Leakage**
- **Functionality Abuses** (ability to use unwanted functionalities)
- **Functionality Compromisal** (DoS Attacks)

In the last years, with the mass diffusion of **Web Services Applications** we assisted to a big diffusion of Client-Server Applications, this because this kind of Software Structure is really flexible and versatile, which the main aim to improve:

- **Usability**
- **Flexibility**
- **Interoperability**
- **Scalability** (compared to centralized, mainframe, time sharing computing)

A **Client** is defined as a requester of services and a **Server** is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration.

Actually this kind of architecture is used to accomplish a large variety of task in many fields:

- **Trusted User DBMS – Database Management.**
- **Specific Trusted Services – Financial/Banking/E-Governement/E-Commerce.**
- **Multi User Data Access to Online Store.**
- **Protected Data Transfer and Chat.**
- **Distributed Function Processing.**

Most Requested **C-S Functionalities** (Elements) can be resumed in these points:

- **The ability to update multiple different DBMSs in a single transaction.**
- **Connectivity to a variety of data sources including flat files, non-relational DBMS, and the mainframe.**
- **Remote procedure calls, remote data access, and message-passing middleware.**
- **Strong Security, public/private key cryptography, digital signatures, digital certificates, SSL, firewalls.**

These are only the basical functionalities that a C-S Software can accomplish, in the last years, with the great diffusion of this Architecture, many more complex tasks are requested, so we can have truly complex and **Exclusively written C-S Applications** which uses **Advanced Proprietary Functionalities**.

These **Advanced Functionalities** in many cases derives from the combination of more basical functionalities (as the one listed previously). But this great scalability, can lead to very complex and **dangerous vulnerabilities**.
This because each Element could be potentially vulnerable to a certain numbers of attacks and in the global context, the complex implications between different Elements could brings to hardly predictable Concatenated Security Leaks.

So under this point of view, an attacker is extremely favorited, because he has many Elements to attack and various potentially dangerous co-implications between elements that could lead to a **Cascade of Attacks**.

From The Security Point of View

From the Security point of view, this kind of applications need to be **Tested** in three steps:

- **Client Side Analysis**
- **Server Side Analysis**
- **Client ↔ Server Interaction**

For each Step there are different types of Vulnerabilities, but is really important to understand that in Client-Server Applications, **Single Side Vulnerabilities** in the **Worst Cases arent't Isolated** from the rest of Application, in other words a badly designed Arch. Could lead for example to dangerous DoS by emulating in malicious ways the *challenge* and *reply* communication.

Let's see now the Essential Security Expectations of C-S Applications.

- **Confidentiality - Informations needs to be kept private**, often Data Transactions includes trade secrets or even sensitive personal informations. Confidential data such as E-Mails, Accounts, Activity needs to be well protected. The application which elaborates these sensitive aspects, needs to be well compartimentalized. These compartiments needs to be well protected, just think to a client browser that stores passwords into some easly accessible location in a weak way, for example the password stred in clear in some temporary file.

- **Integrity - The trustworthiness and correctness of Data**. Sensitive Data transactions needs to be delivered unaltered, . This task is accomplished by compartimentalising the informations by implementing User Authentication. In Certain cases weak Auth. Schemes could lead an attacker to Steal/Modify important user's data.

- **Availability –** Each well trusted or untrusted user needs to have a well define quote of resources, a weak resource check can lead the application to the crash (DoS)

Client Side Security

The Client is the fundamental Interface deputed to talk with the Server, so the first step is to check the security of this Interface. If a Client is accessible via the network or through various mechanisms on the local machine, attackers might be able to connect to that component and gain access to unauthorized resources.

The phrase " if a client is accessible" has many implications, cause **Accessibility** can regard various dangerous situations:

- **Malicious Input Filtering**
- **Weak User Data Storing**
- **Unprotected Temp Files**
- **Vulnerable to Memory Sniffing, this implies also that is Vulnerable to Malware Password Stealing**
- **Proprietary Network Protocols with Custom Encryption**
- **Abuse of Functionality**

A Security Analysis of a Client Application should start from the basical **Input Validation Check**, unfiltered Input Boxes can lead to dangerous DoS and/or Privilege Escalation.

Many times Client Applications uses **Weak Encryption** techniques to store sensitive data as Accounts, History, Processed Data, this kind of weakness can have more variants, such as:

- **Storing Sensitive Data Unnecessarily**
- **Lack of Necessary Encryption**
- **Insufficient or Obsolete Encryption**
- **Originator Validation**

In the same way, Clients leaves trace of the Data Processing in Temporary directories, this could be dangerous cause an attacker could extract Confidential Informations.

The **Memory Sniffing**, is one of the most undervaluated SecurityAspects of a Software, unchecked and/or unprotected Memory Blocks can vanify the presence of a ciphering algorithm, cause an attacker can easly steal the password in clear, or with some Reverse Engineering attempts decode the badly designed Memory Protections.

Often, Client/Server applications uses **Custom Network Protocols** and **Propietary Encryption Algorithms**.

This is a very dangerous situation, because every untested Network Protocols can have Conceptual and Logical errors in their Architecture that can allow

- **Abuse of Functionality**
- **MITM Attacks**
- **Data Manipulation**
- **Session Manipulation**
- **Session Prediction**

- **Session Replay**

Protocol Structure is one of the most important aspects of every C/S Application, because is the Core Link between Client and Server, so any sort of vulnerability into the Protocol, will affect both the Client and the Server Application.

Became clear how many important is the direct Protocol Analysis, to identify the Logical Architecture that uses and the Conceptual Errors that may affect the entire Security of the Application.
The first approach is a classical Protocol Reverse Engineering one, especially designed to gain in-depth understanding of how protocol works, here the first Reverse Engineering Steps to follow:

- **Binary String Examination**
- **Special String Values Examination**
- **Discovering Communication Primitives**

Binary String Examination, Wrongly formatted string request could be really dangerous, can lead to **DoS Attacks** and **Privilege Escalation**. String Examination will help also for the process of localization of Functions that Processes the Packet.

Special String Values, If the Application/Protocol allows the presence of some unchecked Special Char Values, this could be really dangerous because can allow, in certain cases, Code Execution Attacks.

Discovering Communication Primitives, Finding Communication Primitive, will tell us the basilar function used for communication and consequently if function's parameters are correctly managed. In this way we can also reimplement a custom Client that emulates the challenge<=> in malicious ways. A concatenation of vulnerabilities may lead to Abuse of Functionalities.

Badly managed parameters could be a risk, cause they may be susceptible to the most common vulnerabilities, such as:

- **Integer Overflows**
- **Integer Underflows**

On **Text Based Protocols** we can meet the following vulnerabilities:

- **Buffer Overflow (releated to String and Metacharacters)**
- **Format String Vulnerabilities**

After the first Reversing Approach directly centred upon the Structure of the Protocol, became necessary to study the **Interactions** between the protocol and the rest of the system, in particular assumes an high importance the study of:

- **Data Verification**
- **Access to System Resource**

A well designed protocol should check **Trasparently** the **Validity** of **Data Transmitted** or Received, this Verification sometimes could be bugged in its own design:

- **Exploitable Integrity Data Mechanisms**
- **Precomputable Data Verification**

Some badly designed systems of **Integrity Check**, does not have strict **Data Validation** and **Validation Harmoring Criteria**, so an attacker, can find malicious ways to make the Client/Server application to believe that the Data Packet is Trusted and sent by the original server.

In other cases, weakly implemented Integrity Mechanisms could be reversed and dangerously emulated for **Malicious Proprietary Packets Forgery** to inject via MITM Attacks.

Server Side Security

The **Server Side Application** is the second unscindible aspect to Analyse in terms of Security, cause is the **Provider of Requested Services**, its security depends strictly on the Conceptual and Logical organization of the entire C/S Architecture, in many cases we have unseparable security aspects that links the Client Side Security to the Server Side Security, such as the Protocol Implementation.

In the Server Application we have also to analyse the **Data Processig Mecchanisms**. Indeed every Data Element received from the Client needs to be Processed according to the requested functionalities of the application.

This particular aspect could be susceptible of various attacks, strictly linked to the kind of Requests and Data that the Server Receives.
Also for the Server, the first Analysis Step is a Blackbox Reverse Engineering of Incoming data to prevent:

- **Malicious Data Requests**
- **Weak Credential Verification**
- **Anti-Automation**
- **Request Encoding**
- **Content Spoofing**

Malicious Data Requests, are assimilable to the previously seen Input Validation Errors, but in the case of the Server Side Application there are other more dangerous aspects to consider. Dadly filtered Inputs can cause:

- **Denial of Services,** that in the case of the server could have more heavy consequences, if the application is provided by a big Corporation.
- **Buffer Overflows**, Big unckecked volumes of Data could be used to Remote Exploit the Server.
- **Format String Attacks**, As the Buffer Overflows, but in the case of C/S applications is strictly linked to the Protocol Organizzation
- **Anti-Automation**, Malicious Clients could automatize Communication process, and sometimes this is an unwanted feature. In this case the application should implement well designed **Visual Verification Mechanisms**, and consequently these mechanisms should implement strong **Anti Request Encoding Countermeasures**.
- **Brute Force**, the logical input receiving architecture should also verify the presence of Brute Force mechanisms, and sometimes this is not well accomplished because with Dynamic Packet Masquerading the Anti Brute Force mechanism could be overcomed.

In one shot we covered all the possible Malicious Request that a server can receive, and consequently the **Anti-Malicious Requests Mechanism** (and their Vulnerabilities). Other

vulnerabilities could be:

Weak Credential Verification, this aspect could lead to two dangerous situations, such as:

- **Balanced Credential Verification,** the security question involved in this aspect is, *our credential verification is adequate for the resource accessing?* Is truly import to determine whether users should be allowed to access a resource the application provides. This conducts to another security question, *does the application give access to resources that it's supposed to?*
- **Unauthorized Data Access**
- **User Impersonating**

After testing the potential dangerous situation that can be involved into **Ingoing Data Request**, the next step is to adequately sanitize the Server Outgoing Informations. Untrusted Requests needs to be (as previously seen) dropped in the first stage, but for security reasons, also Server Responses should be trasparently checked, this to avoid the following risks:

- **Directory Traversal**
- **Predictable File Location**
- **Configuration Disclosure,** strictly linked to the Predictable File Location.
- **Debugging Informations Accessibility**
- **Verbose Messages,** Sensitive Informations about the Application if not sanitized could lead to Information Disclosure.

The Server Side Application often, works with external applications such as **Database Systems**, or other **Post Processing Units**, so it's of crucial importance to determine:

- **How Data Transactions between Server and Extern Units is Accomplished.**
- **Insight security of the External Application.**

Badly designed systems of data transactions could lead to the already known Cascade Effect that drives the attacker in a series of Cross Applications Request, became clear that is important to correctly compartimentalize and validate each Process Unit involved in the Global Server System, to avoid every kind of Data Disclosure.

Other Important Aspects are involved in the security of a Server Application, such as the correct

- **User Data Protection,** with well designed trusted algorithms.
- **Cryptographyc Session Harmoring,** to prevent any sort of MITM Attacks.
- **Encryption Vulnerabilites**

The entire “packet” of tests needs to be performed in a first time, for separate instances (Client Side and next Server Side) to determine the **Basical Architecture Primitive**, and next is necessary an **Active Analysis** to directly approach the possible vulnerabilities.

Malware Threats

In the last years as you know malware threats have had an **Exponential Increment** in **quantity** and **quality** (diversification). Some year ago, Viruses were only a threat for **End Users**, in a successive period malware became a threat also for **Internal Networks of Companies**. Actually we have seen a great diffusion of malicious applications specifically developped to infect both Client and Server Solutions, especially **Financial Services Companies** and big **Commercial Corporations**.

Let's see the main features of specifically written malicious applications:

- **Client Side** – Personal Sensitive Data Theft – Damages – High Diffusion
- **Server Side** – Massive Theft of Data – Damages – Networked Diffusion

This new kind of malware applications are essentially coded to abuse three already seen problematics:

- **Unauthorized Access to Host System Resources.**
- **Unauthorized Access to Data.**
- **Unauthorized Access to Enduser Information.**

It's interesting to note that Malware targets in the same way both Clients and Users. Indeed if we consider the **Unauthorized Access to Host System Resources** Risk, a malicious application could target Applications, Aevices and Aesources for Damage/Spy purposes both on Client and Server Side. In the same way **Unauthorized Access to Data** could be on Local/Networked Disks or Data Communicated from or by the Client such as: Account Credentials, Financial Credentials, Sensitive Documentation. **Unauthorized Access to Enduser Information** as the previos risk, this last one could work as **Data Miner**.

Now let's consider a **Real World Example**, a malware called **Silent Banker** that in the last months targetted many **Home-Banking Servers** causing many damages both for Servers and Clients (End Users). I decided to report that malware cause it presents many really intersting features that can summarize the major part of aspects previously listed.

SilentBanker, is able to own the Authorization mechanisms and inject itself into the bank transactions in a classical MITM Attack style, between the various functionalities provided by SB we can summarize the following one:

- **Redirect users to an attack-controlled server**
- **Recording keystrokes**
- **Capturing screen images**
- **Stealing confidential financial data**
- **Can silently change the user-entered destination bank account details to the attacker's account details in the middle of a transaction**
- **Duping the bank customer into entering a second authorization password**
- **Intercept authentication traffic before it is encrypted**
- **When the virus installs itself, the web addresses for 400 different banks are downloaded to the victim computer.**
- **The virus steals passwords for file transfer tools, e-mail, and storage.**

Silentbanker sits on the website, and unbeknownst to you it downloads to your system," (cit taken from http://www.canada.com/ottawacitizen/news/story.html?k=86382&id=7f3cf367-e71a-4828-b770-4bcacf1cc39f) said by Huger.

As you can see SilentBanker represent all previously seen problematics, that could affect contemporary Client and Server Structure. By inspecting the attacks accomplished by SB the Security Operator can build an harmored environment, in which every resource is correctly Jailed and Balanced according the Real Effective Necessity.
Here ends this little overview, that is only intended to show the most common problems that could affect a C-S Based Application.

Regards,
Giuseppe 'Evilcry' Bonfa'

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGP Desktop 9.0.6 - Enterprise license

mQENBEd882MBCADm8p57uFAMS5zpe5i3V2J2GfEh8z4iMWtszWrqQQauJipDKxxM
WsdW/85tJ6v+OpBVS+xaT3pg8+g8RQl4heYDhLZ9PJwpImqScJSyNZ2La56SuRgR
BR0uG6C5EFQRK7oQpR63yyXlnIMM4LavLLYtoxuWqUhQ1dMePIWJLlGH5ZpKAiqm
j/DyPzh9BmUFABNKViDHytEaouIt5U/7f9biIs2wRPuwxkT8d3iZ8f0/kNgdM7iQ
5FtOXMCAnbVlOtHUubuuZPPLoBNQwMjpdbmN2X7TLpW0o5x/pxyT0v0AoGUPwOcd
DWox86H5fF7JV+uc7JI0FtgcXkwabdk9a5EJABEBAAG0G0V2aWxjcnkgPGV2aWxj
cnlAZ21haWwuY29tPokBhwQQAQIAcQUCR3zzajAUgAAAAAAgAAdwcmVmZXJyZWQt
ZW1haWwtZW5jb2RpbmdAcGdwLmNvbXBncG1pbWUHCwkIBwMCCgIZARkYbGRhcDov
L2tleXNlcnZlci5wZ3AuY29tBRsDAAAAAxYCAQUeAQAAAAQVCggJAAoJEI4Y3IPu
FM++7WkIAOHWTCOh5LIf+Ak8+sS7bVE4t/PeYGY9tu9DAeD/FxrlSsEOyqk59HiP
p42I2ulPmL24OUFGLnIYhD1oEYAdzZHA/cpFo2T2LzLlB9Btdy0rl3u+51dQv7k0
PtedTHGYSbYoRphP8cRxy4b8mCexhQJTfRHBEVGkeFgLF32u1KRqkiQlJrbJs6FZ
a9jLifIB19Nt0QCAk90wGQUCE0SMYbnTkGeKDQzus1yVitxsQrlKJjXnOQncP8BN
/Za7UsqPeL4hzkkQAdG8kbMuh15WaHLPZbaBSxB9nbL35DwmRgra1swLasPKpO/K
Wveabh9xhzOTy1Pdg3Mr1TiC9jUuRX6JASIEEAECAAwFAkhc7VwFAwASdQAACgkQ
lxC4m8pXrXy6LAf/cRbT2DLiyp578+4lP8HEOXfmtdN/k3laHQoNJoVqpXL8vmt2
Xd3WRfxXAyf4vhYt+yLPjCHJwPr7Nn6E6jKx7wQOb/+xz13vF4k3hvHVhQTDmg4E
m2ST3IywgdAUOJ3YDDGJ1OJixBT1qUAsfm2/OHuPnsg4GN9dxewQUjPQ/h6CHt7I
QQz49jeZlB/bWusiDgHTBK/WCERZegI0EYBQiw6P1RDy5PSssZTMzdh9Ck0CQzl5
VH75M0lVv4zikAVDs3LqmZPulVdv9607EhLVLyMJZ0avwD1sQAfnPs1DaEl0qi1V
bjEEAhtLg63mUITRe6KTScRUkKk9wUcIS+SwDLkBDQRHfPNkAQgArRmCHbZSvq8g
SPF0WUgGo7rCJHvdfaOFUNmRHfFUXDh9e6h8Mtyl9WKJpyMdMhRYQrxXKIujNwEM
TKoOJ2+KBRsjT7XsEWRzMoYv23g3GtZplSKHthGNp1ucoj/pZIrv6mq5cf1/z+/E
5Ev2GUnZainaliQ9LeXbzVmQ3FlNsCe9JkoicU5FtxnyHWPQRJUpa9QV3e/YSH2u
Xp7oDoRycpMGQ0cNCPslg1/RJnMCeVTna6suALr2soHY/ppeVB8ckg7QFLh+WKWq
AVgpBBDpV5ABfvIkRlkqYbVjIUXeubtVwOy44o7JymAQAQo92/oKMtnbybsILnu8
zZOUMX4+UQARAQABiQEiBBgBAgAMBQJHfPNkBRsMAAAAAAoJEI4Y3IPuFM++BIAI
AIaI3S7S1sAYt7vGkzgOm5mEo9RSilMhQRMWLVnk0m2QhohmUmOe6XtRv0QGYdqM
7JonhOzHp6sHzOZUQE39tbECb52KwWedn74vt85Gi/UdTxdSCITOkc2L+5RU+2mX
uzJVLKASob+ZITsjmEqFye+W2bumwSPEy5GaxSxemXJJB3GyMYpiwjpXxvWLd4Y7
nDkRfdqQh5tHuZs0Fsv6UO8ZC1kfTUC79oRlLMNpQnO/jocHvZTI9LSTQ3Vj6rOq
f+moDqp0HfVLcFwf9dTYaqCzhpwlJVzgliPWa+FNrERh3ch/+/FER4kqxunWieJP
xpQuxTS2GI7TEiZ8vgjgoe8=
=MdHE
-----END PGP PUBLIC KEY BLOCK-----