CYBSEC®
Security Systems

SAP PENETRATION TESTING
with sapyto (v1.00)

A CYBSEC-Labs Publication
by Mariano Nuñez Di Croce

April 16, 2009

Abstract

Penetration Testing has become an industry-proven effective methodology to analyze the security level of information systems platforms. However, due to the lack of practical knowledge, fear of service disruption and absence of proper tools, SAP systems have always been excluded from this kind of assessments.

As implementing SAP in a company is such a complex and long process, security settings are often postponed or unattended. Moreover, many of these settings have unsafe values by default. The combination of this two facts results in many insecure SAP platforms, exposed to high risk threats.

This publication describes the use of *sapyto*, an SAP Penetration Testing Framework, which assists security professionals in assessing the security of their SAP platforms. *sapyto* will help them detect existing vulnerabilities and increase the overall security level of the implementation, protecting the company's critical business information.

Table of Contents

1. Introduction

For years, when auditors and security professionals referred to “SAP security”, they were mostly meaning “security of the SAP Authorization subsystem – roles & profiles”. This kind of analysis in mainly focused in analyzing users’ authorizations, with the purpose of detecting and shrinking wide and incompatible privileges that could result in business frauds. This kind of assessment is compulsory for many companies because of regulations like Sarbanes-Oxley (SOX), among others.

While the authorization and Segregation of Duties (SoD) reviews are important, so far the security community has been overlooking an equally significant area in the SAP security matter: the technical security, or “grey area”. This area involves all the technical aspects and components that serve as the base framework for running the business modules and is critical for the security of the platform: a breach in many of these components would also result in the ability of carrying out business frauds, even though the authorization system is absolutely locked down. Therefore, in order to develop a complete assessment, security professionals need to fully understand and analyze these threats.

Penetration testing has become an industry-proven methodology to effectively analyze the security level of companies’ information systems. This kind of assessment provides a unique perspective over the current security state of the information platform.

So far, SAP systems were usually not included in this kind of projects. Fear of interrupting service and lack of knowledge prevented managers (and consultants) from testing this critical infrastructure. The consequence of this is many unsecured SAP implementations, continuously exposed to different kind of attacks.

This document describes the architecture and usage of a robust framework that will assist security professionals in undertaking SAP Penetration Testing projects, strongly focused on the technical security, helping them detect weaknesses and increase the security level of their primary business platform.

2. Methodology and Goals

As with any other type of Penetration Test, it is essential to carry out an ordered and complete assessment. The suggested methodology, widely proven by the international community and compatible with security standards [1] comprises the following primary phases:

1. Discovery
2. Exploration
3. Vulnerability Assessment
4. Exploitation

In an SAP Penetration Test project, the premier goal is usually the achievement of the highest possible privileges over the Production system. This can be accomplished by reaching any of the following access levels:

- SAP Administration privileges at the Operating system level (<sid>adm user) or higher.
- DBA privileges over SAP database schemas or higher.
- SAP_ALL privileges over the Production client or equivalent.

Any of the presented privileges are equivalent: obtaining just one of them will probably result in the complete control over all the other SAP layers as well.

3. Setting up the Assessment Platform

Many of the involved activities imply repetitive tasks that can (and should) be automated with the assistance of particular tools. Furthermore, in order to perform certain analysis, specific software must be used.

The following tools will be helpful while performing SAP Penetration Tests:

- sapyto [2]
- nmap [3]
- r* tools (rsh, rlogin, rexec)
- SQL client tools
- NFS client tools
- SMB client & security tools
- BurpSuite [4]
- Nessus [5]
- john (patched) [6]
- hydra [7]

It is worth mentioning that, despite the fact that many of these tools simplify many parts of the process, it is essential that the security professional performing the assessment has a vast knowledge in general SAP architecture and security concepts, in order to fully understand the existing threats and avoid disrupting the service of the platform under analysis.

4. sapyto: The SAP Penetration Testing Framework

sapyto is the first SAP Penetration Testing Framework. Fully developed at **CYBSEC-Labs**, *sapyto* provides support to information security professionals in SAP platform discovery, exploration, vulnerability assessment and exploitation activities.

sapyto is periodically updated with the outcome of the deep research constantly carried out by CYBSEC-Labs on the various security aspects of SAP systems.

The following sections provide a so far missing guide on how to use this framework, from the installation up to a description of all the plugins and features available in the Public Edition of *sapyto* (v1.00).

4.1. Installation

4.1.1. The SAP RFC Library

While CYBSEC-Labs has developed a fully functional python implementation of the RFC protocol, codenamed CROAKlib, it is not possible to release to the general public.

Therefore, in order to use the plugins that operate through SAPRFC* connectors, the SAP RFCSDK must be downloaded and installed. The RFCSDK is an API released by SAP to allow the development of RFC-enabled software.

This library can be downloaded from the SAP Support Portal, at http://service.sap.com . This will require a valid SAP customer account.

The following RFCSDK releases are fully supported by *sapyto*:

- RFCSDK for Linux 6.40 (Non-Unicode)
- RFCSDK for Linux 7.00 (Non-Unicode)
- RFCSDK for Windows 6.40 (Non-Unicode)
- RFCSDK for Windows 7.00 (Non-Unicode)

For example, in order to download the 7.00 version you will need to:

- Access http://service.sap.com/swdc (login)
- Select "Download" -> "Support Packages and Patches" -> "Entry by Application Group" -> "Additional Components"

- Select "SAP RFC SDK"-> "SAP RFC SDK 7.00" -> Choose your platform -> Download.

Once the library has been downloaded, it must be uncompressed with the SAPCAR utility:

```
$ SAPCAR -xvf file.SAR
```

This will create a "rfcsdk" folder, containing the necessary files to build SAPRFC* connectors.

4.1.2. Installation on Linux systems

Dependencies

This version of *sapyto* should work under any Linux distribution and was successfully tested for IA32 and x86_64 (amd64) architectures.

The following software is required for the base installation of sapyto:

- Python >= 2.5.
- Python development libraries (python-dev).
- GCC runtime and development utilities/libraries

Please verify that all required dependencies have been installed and configured successfully before installing *sapyto*.

Installing

sapyto is shipped as a compressed ZIP file. This file must be uncompressed to an installation folder.

If the SAPRFC* connectors are to be used, *sapyto* needs to be built and linked against the RFCSDK, which can be done with the following command:

```
$ python setup.py build
```

If the *rfcsdk* header files were not deployed to */usr/sap/rfcsdk/include*, you may need to supply the `-I` flag to let the setup script know where to look for them. The same applies for the lib files of course, using the `-L` flag.
For Example:

```
$ python setup.py build_ext -I/foo/bar/rfcsdk/include -
L/foo/bar/rfcsdk/lib
```

```
$ python setup.py build
```

Then

```
$ python setup.py install
```

This will install the required libraries in the system.

Finally, it is necessary to tell the Operating System where to look for the RFC library when trying to load it. This can be done editing the */etc/ld.so.conf* (or equivalent) and adding the path to the library folder, for example:

```
/usr/sap/rfcsdk/lib
```

Afterwards, update the dynamic linker configuration:

```
$ sudo ldconfig
```

Another alternative is to configure the LD_LIBRARY_PATH environment variable.

In order to check if the RFCSDK was successfully installed, run:

```
$ ./sapyto -c
```

You should get that the SAPRFC, SAPRFC_EXT and SAPGATEWAY connectors are available:

```
SAPRFC connector available
SAPRFC_EXT connector available
SAPGATEWAY connector available
...
```

Troubleshooting

If the aforementioned connectors are "`NOT available`", you should check that:

- You have all the packages and software listed in the *Dependencies* section.
- The RFCSDK is installed.
- The *saprfcutil.so* file, located in python packages directory (*/usr/lib/python<version>/site-packages/),* has the correct permissions.
- The RFCSDK directory (*[dir]/rfcsdk/lib/*) is included in the LD_LIBRARY_PATH environment variable or *ldconfig* is updated.

If everything is correctly setup, check if you can access the *saprfcutil* module directly with python:

```
$ python
>>> import saprfcutil
```

If you get an exception like the following, it means that you have to install missing packages:

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: libstdc++.so.5: cannot open shared object file: No such file or directory
```

4.1.3. Installation on Windows systems

Dependencies

This version of *sapyto* was successfully tested for Windows 2000/XP/2003 over IA32 architectures.

The following software is required for the base installation of *sapyto*:

- Python >= 2.5.

Please verify that all required dependencies have been installed and configured successfully before installing *sapyto*.

Installing

The Windows version of *sapyto* is pre-compiled, in order to avoid the installation of many non-native prerequisites on the user system.

sapyto is shipped as a compressed ZIP file. This file must be uncompressed to an installation folder.

Following, open a command line interface and run:

```
C:\sapyto> python setup.py install
```

This will install required modules into the Python library repository.

In order to check if the RFCSDK was successfully installed, run:

```
C:\sapyto> python sapyto –c
```

You should get that the SAPRFC, SAPRFC_EXT and SAPGATEWAY connectors are available:

```
SAPRFC connector available
SAPRFC_EXT connector available
SAPGATEWAY connector available
...
```

Troubleshooting

If the aforementioned connectors are "`NOT available`", you should check that:

- You have all the packages and software listed in the *Dependencies* section.
- The RFCSDK is installed. File *librfc32.dll* is present in *C:\<WINDIR>\system32* (or the search path).
- The *saprfcutil.pyd* file is located in python packages directory (*<DRIVE>\Python<version>\site-packages*\).

4.2. Architecture

In order to understand *sapyto*, it is essential to familiarize with its architecture and terminology. Following, an overview over the different concepts that build *sapyto* are presented.

4.2.1. Connectors and Targets

A *connector* represents a way to communicate with an SAP service or component. Due to the vast number of SAP end-points and protocols, it is necessary to allow the framework communicate by different means to remote SAP systems.

A *target* is, therefore, a configured *connector*. Once a *connector* has been configured, it is already available to be used by the *plugins*.

Currently, *sapyto* supports the following *targets/connectors*:

- **SAPRFC**: Interaction with SAP Application Servers through the SAP RFC protocol.

- **SAPRFC_EXT**: Communication with external servers developed with the SAP RFCSDK library.

- **SAPGATEWAY**: Enables the communications with SAP Gateways.

- **SAPROUTER**: Interaction with SAPRouters.

Of course, the *connector* architecture is fully extensible, enabling the fast development of further means of communicating with SAP components.

4.2.2. Plugins

Plugins implement an particular test or action over a configured *target*. According to their purpose, they are categorized in one of the following groups:

- **Discovery**: Their objective is to discover new targets from the already configured ones. This enables recursive discovery and automatic feedback for other *plugins*.

- **Audit**: These *plugins* perform activities that could be involved in the *Exploration* and/or *Vulnerability Assessment* phases. They obtain further information or perform a check for a particular vulnerability.

- **Exploit**: Their aim is to take advantage of vulnerabilities discovered by the audit *plugins*, enabling the user to escalate privileges or perform security sensitive actions over vulnerable targets.

Besides the *category* it belongs to, each *plugin* is developed to communicate through one of more *connectors*. This allows a *plugin* to perform the same check by different available *connectors*.

Many *plugins* allow the user to specify different options that will influence their behavior when executed. It is possible to obtain more information about available plugins' options in the *Plugins Detailed* section.

4.2.3. Shells

Some of the exploit plugins' purpose is to spawn remote interactive *command shells* over vulnerable targets.

By being able to interact with the remote target through a *shell*, the security professional has the possibility of analyzing the security level of the platform from *a different perspective*, determining further vulnerabilities and analyzing the application of the *security in-depth* principle.

More information about available *shells* can be obtained in the *Shells Detailed* section.

4.2.4. sapytoAgents

sapyto also supports an agent architecture. This model allows the "deployment" of a *sapyto* module (sapytoAgent) after having exploited a vulnerability, which allows the security professional use some *sapyto plugins* (and other tools) in order to assess the security of other systems, *from the controlled one*.

In a similar way as with sapyto *shells,* this functionality allows users to analyze the security level of the platform from a *different perspective*, determining further vulnerabilities and observing the application of the *security in-depth* principle.

More information about available *sapytoAgents* can be obtained in the *sapytoAgents Detailed* section.

4.2.5. Tools

Tools are semi-independent programs that can assist the user in some specific tasks. They can be used from outside of a "scan session".

More information about available *tools* can be obtained in the *Tools Detailed* section.

4.3. Using sapyto

4.3.1. The Console Interface

The console interface allows users to interact with sapyto in text-based consoles. There are two ways of using the console interface: *Interactive* or *Batch*.

Batch mode is the more efficient way to run sapyto in console mode. It is practical use the *interactive mode* for some quick tests and the exploitation phase, but it's not convenient for every day use.

Interactive Mode

Interactive mode can be started just by running "`python sapyto`". This will place the user inside *sapyto* console:

```
sapyto>
```

Here the user can start typing commands. The console has TAB completion (just for commands, not arguments) and history. Keywords are case-sensitive.

Every menu has a `help` command which will explain available options. It is also possible to run "`help <command>`", which will show further information about the specified command.

Target Configuration

In order to start assessing the security of an SAP system, the first action is to define the *targets*. This menu can be accessed through the `targets` command.

```
sapyto> targets
Available connectors: SAPRFC, SAPGATEWAY, SAPROUTER,…
sapyto/targets>
```

The targets menu will show which are the available connectors installed and configured in *sapyto*.
In order to add a new target, the "`add <connector>`" command must be used. For example:

```
sapyto/targets> add SAPRFC
```

Once in the *target* configuration menu, connection parameters can be shown using the "`view`" command:

```
sapyto/targets/add> view
Parameter  Value         Description
=========  =====         ===========
ashost *   127.0.0.1     SAP Application Server host
lang       EN            Language
gwserv                   Gateway service
gwhost                   Gateway host
passwd     SAPYTOPASS    Username password
sysnr *    00            System Number
client     000           Client
user       SAPYTO        Username
```

Required parameters are highlighted with an asterisk (*).
Once all the required parameters have been configured, using the "`set`" command, running "`back`" will add the target for execution. It is possible to view created targets with the "`show`" command:

```
sapyto/targets/add> back
```

```
Added target with ID 0.

sapyto/targets> show
Target ID: 0 [SAPRFC]
SAP Application Server: 127.0.0.1
System Number: 00
Client: 000
Language: EN
Username: SAPYTO
Password: SAPYTOPASS

sapyto/targets>
```

Running "`back`" returns the user to the main menu.

Plugin Configuration and Selection

After *targets* have been successfully configured, it is time to specify which *plugins* are to be run in the session.
In order to do this, the "`plugins`" command must be in the main menu:

```
sapyto> plugins
sapyto/plugins>
```

Here it is possible to see all available *plugins* with the "`list all`" command.
Also, running "`list <type>`" (e.g: list discovery, list audit, list exploit, etc) will show the available plugins of that *type.* For example:

```
sapyto/plugins> list discovery
discovery plugins:
ping        Checks if remote RFC server…     checkGwMon
Detects if the SAP Gateway can …
...
```

It is possible to analyze which *plugins* are compatible with configured *targets*, you running "`list <type> compat`".

Some *plugins* have options and can be configured. This can be accomplished by executing "`<plugin type> config <plugin name>`" (e.g: discovery config checkRFCEXEC). This will send place the user in the plugin configuration menu, where he can view and define options through "`view`" and "`set`" commands.
For example:

```
sapyto/plugins> discovery config checkRFCEXEC
sapyto/plugin/checkRFCEXEC> view
```

```
Parameter     Value            Description
=========     =====            ===========
regTpnames*    rfcexec,execute  List of tpnames to try.

sapyto/plugin/checkRFCEXEC> set regtpnames id1,id2
sapyto/plugin/checkRFCEXEC> view
Parameter     Value            Description
=========     =====            ===========
regTpnames*   id1,id2          List of tpnames to try.

sapyto/plugin/checkRFCEXEC> back
```

After configuring all the required plugins, it is *necessary to add them to the execution pool*. This is done by running "`<plugin type> <plugin>`", e.g:

```
sapyto/plugins> discovery checkRFCEXEC
```

Note that many plugins can be added at once, separated with ",". Special options "`all`" and "`!`" can also be used.

Running "back" returns the user to the main menu.

Starting the Scan Session

Starting the session is easy:

```
sapyto> start
```

All selected *plugins* will be executed against configured (and compatible) *targets*.

If any of the configured *plugins* has not been executed, it is probably because it is not compatible with the configured *targets*. This behaviour can be analyzed by running *sapyto* in "debug" mode, starting it with the "`-d`" switch.

Batch Mode

Batch mode is the more efficient way to use sapyto. There are three different kinds of files that can be fed to *sapyto* for batch processing:

Type	Argument	Usage
Target	-t <file.stf>	Specify targets for the session
Scripts	-s <file.ssf>	Configure and select plugins to use for the session
Output	-o <file.sof>	Configure and select output plugins only

These files contain one "console command" per line, just as if they were typed in the *sapyto* console.
Self-explanatory examples are located in the *scripts* directory.

4.3.2. The Graphical User Interface

The GUI is only available in *sapyto Advanced Edition*.

4.4. Plugins Detailed

This section describes the purpose and configuration of available *plugins* in *sapyto Public Edition v1.00.*

4.4.1. Discovery Plugins

ping

Description
Detects available SAP Application Servers and external RFC servers by sending RFC ping packets.

Connectors
SAPRFC, SAPRFC_EXT

Options
None

Result
Notifies if the selected target is reachable through RFC.

checkGwMon

Description
Detects if it is possible to connect to the Monitoring service of SAP Gateways. This service allows remote administration of the component and permits the discovery of other connected systems.

Connectors
SAPGATEWAY

Options
None

Result
Notifies if the selected SAP Gateway has the Monitoring service enabled.

checkRFCEXEC

Description
Bruteforces possible and common names (tpnames) under which the external RFC server *rfcexec* may be registered at an SAP Gateway.

Connectors
SAPGATEWAY

Options

Name	Type	Required	Description
regTpnames	List	Yes	List of tpnames under which rfcexec may be registered at an SAP Gateway.

Result
If detected, a list with the discovered tpnames of *rfcexec* servers.

saprouterSpy

Description
Performs internal hosts port scanning through SAProuters.

Connectors
SAPROUTER

Options

Name	Type	Required	Description
type	Choice	Yes	The target type. Possible entries: *file* | *iprange*
targets	String	Yes	The targets of the scanning. If *type* is set to *file*, *targets* points to a filename with one target per line. If *type* is set to *iprange*, *targets* specifies the hosts to scan.
mode	Choice	Yes	Scanning mode. Possible entries: *sap* | *normal* | *full* | *<a port range>*

			sap mode scans SAP related services. *normal* mode scans *sap* mode + some administration and common ports. *full* mode scans the 65535 TCP ports. It is also possible to specify *a port range*, i.e.: 21-25,80,443
createTargets	Boolean	Yes	Specifies if targets are created automatically when associated connectors are detected. Possible entries: *true* | *false*

Result
A list of detected open and reachable ports in configured targets, behind the SAProuter.
If the *createTargets* option is set to "true", and detected services have *sapyto* connectors associated, new targets are created.

4.4.2. Audit Plugins

bruteLogin

Description
Brute-forces user/password combinations in order to identify access credentials to the SAP system. Please beware that SAP implements user locking for protecting from brute-force attacks.

Connectors
SAPRFC

Options

Name	Type	Required	Description
type	Choice	Yes	The bruteforce type. Possible entries: *defaultUsers* | *wordlist* If *type* is set to *defaultUsers*, only well-known default credentials will be tested (SAP*,DDIC,SAPCPIC). If *type* is set to *wordlist*, user and passwords wordlists can be specified.
userWordlist	String	No	The username wordlist filename. Records must use the following format: *client:userid:[passwd1,passwd2,..]* If no *pwdWordlist* was specified, it is possible to specify a list of passwords to try for each user.
pwdWordlist	String	No	The password wordlist filename. The file contains one password per line. If specified, each user from the *userWordlist* file is

			tested with all the defined passwords from *pwdWordlist*.
tryUserAsPwd	Boolean	No	Specifies if the username is to be tested as the password. Possible entries: *true* | *false*

Result
A list of discovered credentials.

getClients

Description
Enumerates available clients in the target SAP system.

Connectors
SAPRFC

Options

Name	Type	Required	Description
initialClient	Integer	Yes	The initial client from which enumeration will run.
endClient	Integer	Yes	The last client up to which enumeration will run.

Result
A list of the discovered clients.

getDocu

Description
Obtains the documentation of installed functions in external RFC servers.

Connectors
SAPRFC_EXT

Options
None

Result
The installed functions on a remote external RFC server, plus the optional documentation about its interface (parameters and tables).

registerExtServer

Description

Attemps to register an external RFC server at an SAP Gateway, in order to verify the configuration of the *gw/sec_info* and *gw/reg_info* files.

Connectors
SAPGATEWAY

Options

Name	Type	Required	Description
tpname	String	Yes	The program name to use when trying to register.

Result
A message indicating if it is possible to register the external RFC server at the SAP Gateway.

sapinfo

Description
Obtains valuable information from remote SAP Application servers and external RFC servers.

Connectors
SAPRFC, SAPRFC_EXT

Options
None

Result
Information about the remote system, including operating system, database engine and host, SAP system ID, version information, etc.

saprouterList

Description
Obtains information from remote SAProuters.

Connectors
SAPROUTER

Options
None

Result
Information about the remote SAProuter, including connection tables and operating system environment information.

4.4.3. Exploit Plugins

callback

Description
Implements the RFC callback attack [8]. This can be used to highjack connections to RFC servers and perform callbacks to the SAP Application Server, executing external OS commands over the system.

Connectors
SAPGATEWAY

Options

Name	Type	Required	Description
spawn	Choice	Yes	Type of object to spawn after exploitation. Possible entries: *shell* | *rfcCall* If *spawn* is set to *shell*, a new shell object will be created to run external OS commands. If *spawn* is set to *rfcCall*, an *rfcCall* interface will be created to perform arbitrary RFC calls.
tpname	String	Yes	The name of the original external server.

eviltwin

Description
Registers an RFC external server at an SAP Gateway to perform Eviltwin [8] attacks.

Connectors
SAPGATEWAY

Options

Name	Type	Required	Description
tpname	String	Yes	The name of the original external server.

Commands

Name	Description
help	Display help about available commands.
register	Register at the SAP Gateway.
disconnect	Disconnect from the SAP Gateway.

back	Go back to the previous menu.

gwmon

Description
Connects to the SAP Gateway Monitor, allowing the execution of remote administration commands.

Connectors
SAPGATEWAY

Options
None

Commands

Name	Description
help	Display help about available commands.
connect	Connect to the SAP Gateway Monitor
getClients	Get connected clients (useful to get tpnames).
getSecurityInfo	Get the content of reg_info and sec_info.
getInfo	Get information about the SAP Gateway.
shutdown	Shutdown the SAP Gateway.
disconnect	Close connection with the SAP Gateway Monitor.
back	Go back to the previous menu.

rfcShell

Description
Creates an authenticated RFCSHELL object, which permits the execution of external OS commands over an SAP Application Server.

Connectors
SAPRFC

Options
None

rfcexec

Description
Creates an RFCEXECSHELL object, which allows the execution of OS commands and file read/write through an *rfcexec* server.

Connectors
SAPRFC_EXT

Options
None

saprouterAgent

Description
Creates an *saprouterAgent* object, which allows the routing of *sapyto* and other tools traffic to the internal network.

Connectors
SAPROUTER

Options

Name	Type	Required	Description
bindAddress	String	Yes	IP Address of the interface to bind the agent local endpoint.
bindPort	Integer	Yes	Port on which the agent local endpoint will be listening for requests.

stick

Description
Blocks connections to an external RFC server working in registration mode.

Connectors
SAPRFC_EXT

Options

Name	Type	Required	Description
useBlockingCall	Boolean	Yes	Indicates if the plugin uses a special RFC function to block connections in multi-threaded servers. Possible entries: *true* | *false*

Commands

Name	Description
help	Display help about available commands.
connect	Connect to the external RFC server
disconnect	Close connection with the external RFC server.
back	Go back to the previous menu.

4.5. Shells Detailed

RFCSHELL

Description
Useful for running external commands over SAP Application Servers through RFC. If the target system is running in a Windows platform, it is also possible to execute arbitrary system commands.

Commands

Name	Description
help	Display help about available commands.
run	Run an operating system command (Windows only).
runExt	Run an SAP external OS command.
back	Go back to the previous menu.

RFCEXECSHELL

Description
Useful for running operating system commands and reading/writing files through *rfcexec* servers.

Commands

Name	Description
help	Display help about available commands.
run	Run an operating system command.
runS	Run an operating system command. Stealth. No output.
readFile	Read a remote file.
writeFile	"Upload" a file to the server (limited).
back	Go back to the previous menu.

4.6. sapytoAgents Detailed

saprouterAgent

Description
The saprouterAgent is “deployed” by the saprouterAgent exploit plugin. In this plugin, it is necessary to configure the agent local end-point.

The *saprouterAgent* is a virtual agent, meaning that no code is uploaded to the remote target. The agent works by creating a local SOCKS v4 proxy, which is coupled with a sapyto SAP NI protocol library, effectively translating traffic from local applications to the SAProuter communication protocol. This enables the routing of local applications, through remote SAProuters, to internal services located on the target network.

Commands

Name	Description
help	Display help about available commands.
startProxing	Start proxying traffic through saprouterAgent.
back	Go back to the previous menu.

4.7. Tools Detailed

getPassword

Description
Implementation of the SAP obfuscation/de-obfuscation algorithm to uncover passwords captured sniffing RFC traffic.
The *Public Edition* version only allows the decryption of traffic between SAP Applications Servers and external RFC clients developed with the RFC SDK.

Usage

-h Print help message.
-d Password string to de-obfuscate.
-o Clear-text password string to obfuscate.

De-obfuscation example
getPassword -d "e5 bf 21 67 00 61 79 68 77 68"

Obfuscation example
getPassword -o sapytopass

5. Conclusions

As it has been presented throughout this document, there is a concrete need of assessing the security of SAP systems from a different perspective. While authorizations review is still fundamental, overlooking the technical security aspects can result in even more dangerous threats.

By default, many configuration settings are not as secure as they could be, which combined with implementation and administration mistakes leave platforms exposed to external attacks that could affect the confidentiality, integrity and availability of the fundamental business information.

In order to assist security professionals in this intricate scenario, *sapyto* provides a solid framework with many features to analyze the current security level of customers' SAP implementations, checking for common misconfigurations as well as for the more complex vulnerabilities.

6. References

[1] OSSTM - http://www.isecom.org/osstmm/
[2] sapyto – http://www.cybsec.com/EN/research/sapyto.php
[3] nmap - http://www.insecure.org
[4] Burp Suite – http://www.portswigger.net
[5] Nessus – http://www.nessus.org
[6] John The Ripper - http://www.openwall.com/john/
[7] hydra - http://freeworld.thc.org/thc-hydra/
[8] Attacking the Giants: Exploiting SAP Internals - www.cybsec.com/upload/CYBSEC-Whitepaper-Exploiting_SAP_Internals.pdf