SAdoor - A non listening remote execution server

Claes M Nyberg <cmn@darklab.org> <md0claes@mdstud.chalmers.se>

2003-03-28

Abstract

In order to remotely control another computer we normaly use some well known service like SSH which listens for connections from clients on a default port. On dedicated servers like mail or file servers, is it mainly used by administrators for remote control.

Althought the service might not be usable by anyone it often reveals alot of information about the system, and it might contain a bug allowing arbitrary commands to be run on the machine for anyone allowed to connect.

There are even systems like IDS (Intrusion Detection System) sensors which uses an unaddressed network card to gather information about the network traffic, making it impossible to run a listening service. This paper describes how (to use SAdoor) to remotely control a (UNIX) machine without any listening service.

Contents

Chapter 1

Introduction

The idea is pretty simple, SAdoor listens for a set of packets on a specific NIC and when the first packets (key-packets) has arrived, the last packet (command-packet) is accepted, which contains a shell command for SAdoor to run, or information about how to established a connection.

The sadoor-command is symmetriclly encrypted (Blowfish) using a 448 bit key which is shared between SAdoor and the clients. The client program sash(1) takes care of sending the key-packets and the command-packet, making the procedure as transparant as possible for the user/administrator.

The information required by the client to run a command or establish a connection on the host running SAdoor is stored as an image of the current SAdoor-configuration in a local database, which also is encrypted.

SAdoor is a remote administration tool, so access should only be provided to the people sharing the root account.

Chapter 2

Packet Configuration

The configuration of the awaiting packets is the main thing to focus upon before attempting to run SAdoor. The requirements are small, and only a destination address and a protocol is required in each packet. It is however possible to mimic almost any type of packet belonging to a "real" connection if the optional settings is used.

To prevent "real" packets to trigger SAdoor, it is possible to set a timeout value that resets the list of received packets, forcing all the packets to be sent again.

Supported protocols and their optional settings

IPv4

- Source address
- Type Of Service
- Identity
- Time to live

TCP

- Destination port
- Source port
- Control flags
- Sequence number
- Acknowledge number
- Data

UDP

- Destination port
- Source port
- Data

ICMP

- Type (currently only 0 and 8 (echo reply/echo request))
- Code (must be zero since type is limited)
- Data

One field of particulair interest is the IP TTL (Time To Live) field. This field makes it possible to limit the radius of computers which can produce valid SAdoor packets.

If this field is set to it's max value (255) only the computers residing on the same segment as the machine running SAdoor will be able to produce valid sadoor-packets. It is also possible to configure the client program to add a number to a required TTL value to compensate for reduction in transit (we normaly say that this is the number of hops away from a computer.[1]), as well as a default value for any unused field.

In order to mimic "real" packets, the payload/data can be set in any packet, making it possible to add application protocol headers (with or without data) like HTTP or SMTP.

The beginning of the payload in the received packet is then compared against the expected one and any trailing data is ignored. This is useful if we for example want to create a valid packet from a HTTP GET request on a certain file but we don't care about the rest. One thing worth mentioning is that any data set in the command-packet also works as an offset to the encrypted sadoor-command.

2.1 Packet grabbing

There are two ways for SAdoor to grab the packets, promiscous or non-promiscous mode.

In non promiscous mode, only the packets with a destination address matching any address of the NIC used for grabbing packets is received by SAdoor.

The promiscous mode makes the NIC grab any packet passing through it, making it possible to send commands with packets going to a completely different host. This is the mode to use when SAdoor is running on a machine that should not be or isn't visible on the network (like an IDS sensor or a bridge).

[1]The TTL field is acctually reduced by one when it enters a router and then by one for each seconds that it stays there, but it hardly ever stays more than a second.

Chapter 3

SAdoor commands

The sadoor-command resides in the payload of the command packet either by its own or after/inside any other data and has the following structure:

```
struct sacmd {
    uint8_t iv[8];          /* Initial vector */
    uint16_t length;        /* Total length including data */
    uint16_t code;          /* Command code */
    uint8_t data[CMD_MAX]; /* Command data */
};
```

The initial vector (IV) is a timestamp (seconds and microseconds in GMT) used for encryption of the sadoor-command (everything except the IV itself) in CFB (Ciphertext Feedback) mode, making two equal commands enciphered to two different ciphertexts.

The array contains the actual command and its type depends on the command code which is one of following:

3.1 Run Command

In this case, the data is simply a string that is passed on to system(3) by SAdoor, making it possible to run arbitrary commands without the need of establishing a connection.

Since the the command itself can be part of any larger chunk of data as long as the data prepended (offset to the sadoor-command) is set in the configuration of the command packet, it is possible to prepare a file that is fetched through an unencrypted channel like FTP which at the same that it is transfered secretly passes the command on to SAdoor.

3.2 Connect command

This is the default command sent by the client program, and it tells SAdoor to connect-back to the client.

The array data is then interpreted like this:

```
struct connect_cmd {
    uint32_t ip;
    uint16_t port;
};
```

Where ip and port is the (IPv4) address that the client waits for a TCP connection from SAdoor on.

3.3 Accept command

Passive mode, this command tells SAdoor to listen on a port selected by the client, which is useful if the client is behind a firewall, the data array is in this case interpreted as the following structure:

```
struct accept_cmd {
    uint16_t lport; /* Local port to listen on */
    uint32_t cip;   /* Client IP */
    uint16_t cport; /* Client port */
};
```

Before the client sends the command, an address is bound to the socket later used for a connection attempt to SAdoor. When the client connects, SAdoor compares the address against the one received in the command-packet, and aborts if it does not match.

3.4 NULL command

A NULL command is a command-packet without a SAdoor command, i.e. a packet with a payload that equals the payload set in the configuration of the command packet.

This is really dangerous option which of course is turned off by default since it only requires knowledge of the packet configuration, but it makes it possible to tell SAdoor to run a predefined command without the need of transmitting any data at all. This is as stealthy as it gets, with a wisely configured packet configuration it is as close to impossible as it will ever be for anyone to notice this backdoor (on *your* machine).

As mentioned above, SAdoor can be configured with a timeout value for the required packets. This comes really handy here, since we could make the required packets fit common connections made to some or all of the hosts on the same segment if promiscous mode is used for grabbing. Instead of relying on the packet combination only, we rely on very common packets for a specific amount of time.

Imagine that the timeout is set to a short amount of seconds, and that the packets match a combination of common requested HTML pages. All you have to do is to fire up your browser and request the pages in the right order in the range of the timeout value to run the command.

Or even better, write a program that make the requests for you, and run it on different hosts in the right sequence.

3.5 Replay Attacks

If someone sniffs the network and figures out the packet configuration (which probably is quite easy if the same packet configuration is used for two commands), he can not run arbitrary commands since that requires knowledge about the key, but he could resend the same packets again, repeating the command. This is not good, and could be avoided by enabling replay protection in the SAdoor configuration file.

When replay protection is enabled, SAdoor refuses to run any command with a timestamp less than or equal to the previous (the first IV is compared against the time that SAdoor where started).

Since a clock skew might produce the same affect, SAdoor keeps a list of the last 100 timestamps received in order to make a somewhat accurate log entry. If the received timestamp is less than or equal to the previous and found in the list, it is logged as a replay attack attempt. If it is not found in the list, but it is greater than the first in the list, it is assumed to be a clock skew.

Note that if an attacker changes the timestamp to be ahead of time, the command code will decrypt to rubbish and SAdoor will refuse to perform the requested action. It is also not possible to perform a DOS attack by injecting future timestamps since the command code is checked before any replay protection take place.

Chapter 4

Connection established

When a connection is established using either the CONNECT or ACCEPT command described above, SAdoor generates a new key, the session key to be used during the connection. The key is then sent to the client encrypted with the shared key together with two randomized IV's, one for decryption and one for encryption in OFB (Output FeedBack) mode.

When the new key has been exchanged, a PTY is allocated and the program set in the configuration file (/bin/sh if unset) is started on the slave/client side.

To make things easier, it is possible to transfer files in the current connection. File transfer mode is entered by the presence of an escape sequence in the data received from the client. Only one file at the time is allowed for transfer, and the output of any command running is delayed during the transmission.

Chapter 5

The tools

5.1 Server side

sadoor

The daemon himself. Listens for packets on a specific NIC.

Commands mentioned above can be turned on or off. Sensitive information like received packets and commands can be written to a "private" file while "normal" log messages can be written using any of the common facilities and priorities.

mksadb

Creates an image out of the current SAdoor configuration. Since the output is unencrypted, it should be erased as soon as it has been transfered to a client in a secure channel.

5.2 Client side

sash

The client to use for sending the sadoor-packets.

If the packets should be generated elsewhere like in a real connection, sash can be used for dumping the encrypted payload required in the command packet as well as listening for a connection or attempting to connect to a host running SAdoor without sending any sadoor-packets.

sadbcat

Creates/Updates/Appends a local (encrypted) database of SAdoor hosts.

Instead of having one large file with all hosts, it is possible to configure sash to read a specific file for a certain host.

sadump

Dumps information from the database created with sadbcat or mksadb in either raw or human readable form.

Bibliography

[1] TCP/IP Illustrated Volume 1 The Protocols
By W. Richard Stevens (1994)

[2] UNIX Network programming
By W. Richard Stevens (1998)

[3] Advanced programming in the UNIX environment
By W. Richard Stevens (1993)

[4] Handbook Of Applied Cryptography
By A. Menezes, P. van Oorschot, and S. Vanstone (1996)