

SLAD 2 Developers and Administrators Guide
Version 1.1.1 2006

Copyright:

Lukas Grunwald (info@dn-systems.de)

Christian Schmidt (info@dn-systems.de)

Meike Reichle (info@dn-systems.de)

DN-Systems Enterprise Internet Solutions GmbH, Hornemannstr. 11-13, 31137 Hildesheim

Tel: +49/5121/28989-0, Fax: +49/5121/28989-11

Date: August 03, 2006

Contents

1. What is SLAD	5
2. How to communicate with SLAD	7
2.1. Starting the daemon	7
2.2. Stopping the daemon	7
2.3. Getting information from the daemon	7
2.4. Starting jobs	8
2.5. Optional parameters	8
3. Plugins	9
3.1. chkrootkit	9
3.2. clamav	9
3.3. john	9
3.4. lsof	9
3.5. tiger	10
3.6. tripwire	10
3.7. Snort	10
3.7.1. Snort-Installation	10
3.8. LMSensors	11
3.9. LogWatch	11
3.10. TrapWatch	12
4. SLAD Nessus integration	13
4.1. slad_init.nasl	13
4.2. slad_run.nasl	14
4.3. slad_fetch_results.nasl	14
4.4. Additionally used includes	14
5. Expanding SLAD	15
5.1. General format of a plugin description	15
5.2. The <i>plugin.xml</i> format	15
5.3. A sample <i>plugin.xml</i>	16
5.4. Verifying the <i>plugin.xml</i>	17
5.5. Using the new plugin with Nessus	17
5.6. Golden rule - don't use dynamically linked files	18
5.7. Golden rule - separate audit tools from the system	18

A. John the Ripper	19
A.1. Modification done by BOSS Team to John the Ripper	19
A.2. Configure John the Ripper's wordlists	19
A.3. Configuring John the Ripper's rule file	20
A.4. Ordinary commands	20
A.5. Commands which may use character classes	20
A.6. Character classes for use in the above commands	21
A.7. New John commands	21
B. Tripwire rules	23
C. Links and further information	25

1. What is SLAD

The Security Local Auditing Daemon, or short SLAD, is the local component for network security scans done by the BOSS project.

In principle the SLAD is a daemon that can run any given program as a demon, capturing that program's output on the standard output and standard error streams. This output is stored in memory until requested by the user.

For ease of extensibility and maintenance, the daemon reads the information about plugins (programs to be run on the local machine for auditing purposes) on startup from XML-based configuration files.

2. How to communicate with SLAD

The SLAD executable, `/opt/slad/bin/sladd`, contains both the daemon and a command-line based frontend to it. When invoked with wrong or without any parameters, it displays the following helptext:

```
Usage: sladd [OPTION...]  
-p, --pluginpath=/opt/slad/plugins      plug-in directory  
-l, --listenport=17002                  the port the daemon will listen on  
-d, --daemonize                          run as daemon  
-s, --show=plugins|jobs|<resultid>     display status  
-r, --run=pluginid                       run a job  
-q, --quit                               terminate the daemon  
-h, --help                               display this help
```

The meaning of this option is described in detail in the following sections.

2.1. Starting the daemon

While the daemon can be explicitly started with the `-d` or `-daemonize` command line options, this is not necessary for operations. If a command that requires a running daemon is given and none is found running, one is automatically forked.

2.2. Stopping the daemon

A running daemon can be stopped with the `-q` or `-quit` command line options. This will terminate all currently running processes as well as remove all stored output from finished processes.

2.3. Getting information from the daemon

There are three kinds of information that may be requested from the daemon with the `-s` or the `-show` command line option. These are:

- Known plugins with `plugins`
- Running and finished jobs with `jobs`

-
- Logged process messages with the id of a finished job

The output of the `plugins` and the `jobs` are machine parsable colon-separated lists. The output of `-show=plugins` is meant for a frontend to create a list of options for a user, while the output of `-show=jobs` is meant to determine which jobs are ready and have output ready to be read.

2.4. Starting jobs

Jobs are started with the `-r` or `-run` options, with the id of a job as a parameter.

2.5. Optional parameters

The parameters `-p/-pluginport` and `-l/-listenport` are optional and should only be used for testing or unusual installations.

The former one changes the directory in which the `sladd` searches for plugins. The default is `/opt/sladd/plugins`.

The latter one changes the port to which `sladd` binds to in order to communicate with the command line client. Note that `sladd` tries to use the IP V6 localhost address, `::1`, as default, and falls back to the IP V4 localhost, `127.0.0.1`, if it is not available.

3. Plugins

As shown above, the sladd is just a program to run other programs from inside a daemon and provide an unified interface to their output. The current package of SLAD for BOSS contains the following plugins:

3.1. chkrootkit

The chkrootkit package is a tool to locally check for signs of installed rootkits.

3.2. clamav

The clamav plugin provides a GPLed virus scanner for linux. The options include scanning with or without archive (.zip, .tar.gz, etc) scanning, and removing infected files or putting them into quarantine.

3.3. john

John the ripper is a fast password cracker. This tool is meant to find weak user passwords, which could compromise system security. It comes with three options:

Fast crack mode: In this mode, John only tries the usernames and derived words against the hashed passwords.

Dictionary mode: In this mode all words from the installed dictionary are tried to attack the hashed passwords.

Full crack mode: This slowest mode tries all words from the dictionary, as well as rules generated variations of these, against the user's passwords.

For more details on John the Ripper, including customization and changes done by the SLAD team, see Appendix A.

3.4. lsof

The unix system utility lsof simply shows a list of files currently open on the system and which program uses them. This can assist an administrator to find unusual activity on the system.

3.5. tiger

The tiger suite is a package to analyze the host's security. Out of the many checks the suite can perform four groups have been created. These are:

Users: The users check covers accounts, checks for mail aliases, ftp login users and the like.

Permission: This selection checks users and group access permissions on device nodes, logfiles and other important files and directories.

Config: This script checks for weaknesses and mistakes in common system and application specific configuration files.

System: The system check looks for open deleted files, processes that are waiting for incoming connections, and other "unusual" things.

Full system check: This runs all of the above checks.

3.6. tripwire

Tripwire is an open source file integrity checker. It initially stores hashes of system files in a database for comparison on subsequent runs. Modifications performed by a potential intruder can be easily spotted this way.

The default installation of tripwire contains a rule set for Debian. For details on how to adapt these to a different operating system or distribution, see Appendix B.

3.7. Snort

Snort is a network intrusion detection and prevention system that provides real time traffic analysis and packet logging on IP networks. It is capable of detecting a large number of attacks such as buffer overflows, stealth port scans, CGI attacks, SMB probes or OS fingerprinting attempts by doing both protocol analysis and content checks. Once an attack has been detected Snort is also capable of counteracting them by dropping the according connections. The SLAD-Plugin selects from a MySQL Database all relevant Snort-Messages, and sends it to the Management-Plattform.

3.7.1. Snort-Installation

To use this Snort-Plugins, the Snort needs to be installed with MySQL-Support, this could be done with Debian by using the APT-GET tool.

Answer for the Configuration with mysql to use the snort-mysql database. For the Hostname use your MySQL-Server Host where the SLAD-Plugin collects to. In the most cases this is 127.0.0.1, but you can use any other host here. Then use the database you want to

use for logging, in most cases this will be "snort", you must install mysql first, and create the database via "mysql create snort" and set the permissions first. For further information consult your mysql-manual.

```
# mysqladmin create snort
# apt-get install mysql-server snort-mysql
# zcat /usr/share/doc/snort-mysql/create_mysql.gz | mysql snort
```

After you installed Snort, you need to change the local-plugin configuration. This could be found at "/opt/slاد/plugins/snort/snortconfig"

```
#!/bin/sh
SNORTDBPW="changeme"
MYSQLHOST="localhost"
MYSQLUSER="snort"
MYSQLDB="snort"
SID="0"
```

You can test the configuration by fetching the local-events by running:

```
# ./getsnortevents.sh
```

3.8. LMSensors

This fetches the events from your Hardware-Monitoring, if someone opens the chassis of the server, and your server-mainboard, Management-System supports Hardware-Sensor Logging a alert will be shown to proof the physical incident on the SLAD Managed Server, this Features is supported from the most mid-range Server-Boards like Intel BX440 and newer.

3.9. LogWatch

Logwatch extracts evens from the System log, the Syslog-Files present at "/var/log". All important informations like login-Users, SSH and PAM Sessions, etc are filtered, aggregated, to the calling SLAD. Three different Detail-Level are supported:

-low Returns logfile values in a low detail level highest aggregation.

-medium Returns logfile aggregation in a medium detail level.

-high Full and lowest aggregation level of logfile-values.

See the Link at the end of this Document for Customizing the Logwatch system, if necessary.

3.10. TrapWatch

Trapwatch a special Version of Logwatch, listens on SNMP Hardware-Traps. The Simple Network Management Protocol (SNMP) is the most common protocol for managing all kinds of network devices and is implemented in almost all currently available network devices. An SNMP trap is a message sent out by a network device to report an incident such as loss of link, failed authentication attempts etc. Trapwatch catches these messages and puts them into the report, if someone e.g. unplugs a Machine from a Ethernet Switch, or plugs his own Notebook in, and the Switch supports the Generic IF-MIBs, this is useful to detect, and escalate it. Also support for Netscreen Firewall-Traps, HP-Procure Switches, Cisco Hardware is installed out of the Box. If non-standard MIBs are in use, customisation is necessary.

To enable trapwatch, you need to install a SNMP-Trap Handler, that puts the TRAP-Results into a Syslog-File, if you use Debian you can use the SNMP Trap Format Package.

```
# apt-get install snmptrapfmt
```

After the Service is installed, you will get all new Traps from the Box, SNMP-Traps need to be correct configured in your Network-Hardware enviroment, it is highly advised to test your setup. To test the SNMP-Trapwatch Feature, you can call the Trapwatch Subsystem manually via:

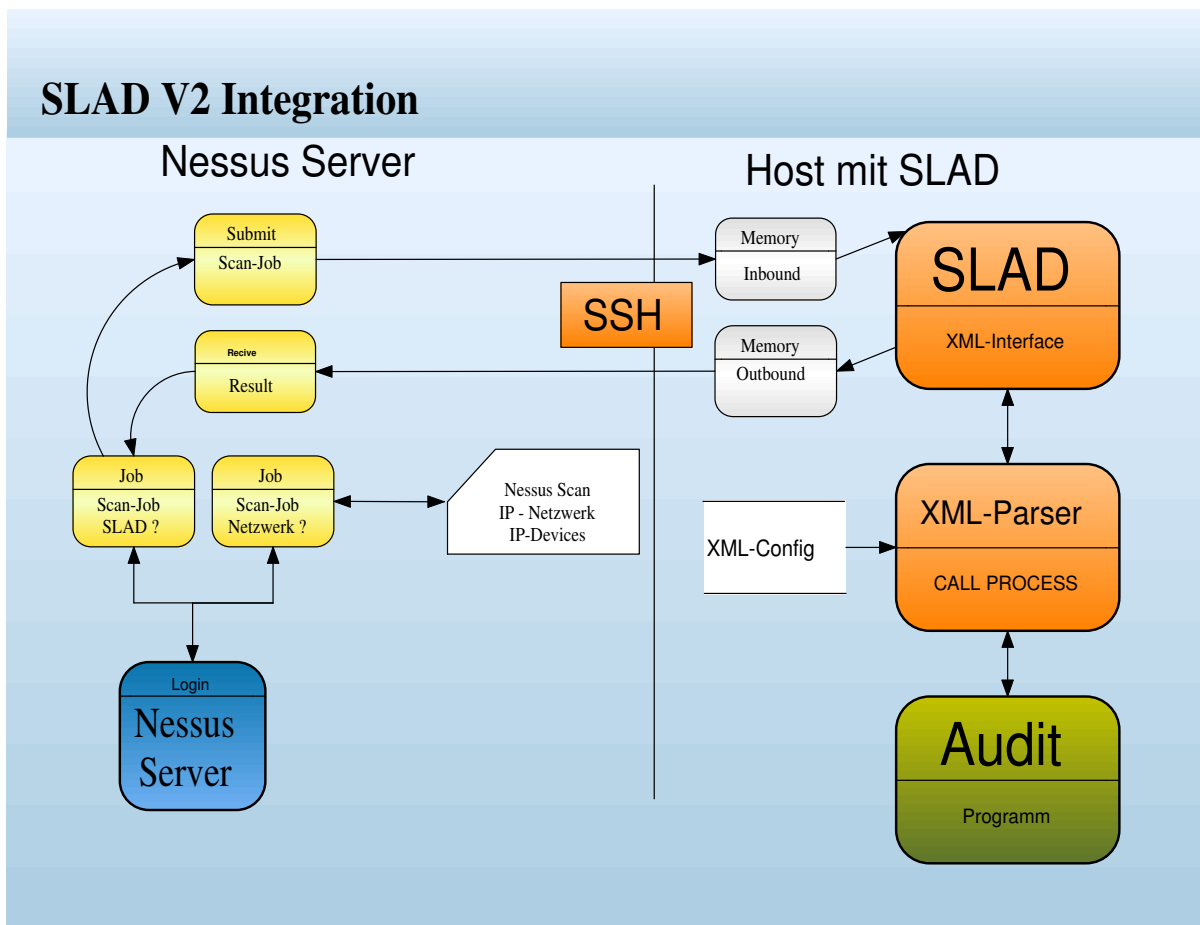
```
# /opt/slad/plugins/trapwatch/trapwatch.sh --high
```

The Result should look like:

```
I 08/18/06 12:28:27 ports: port C4 is now off-line
I 08/18/06 12:28:30 ports: port C4 is now on-line
I 08/18/06 12:28:32 ports: port C4 is now off-line
I 08/18/06 12:28:49 ports: port B4 is now on-line
I 08/18/06 12:29:10 ports: port B4 is now off-line
2006-08-18 14:31:25 [Root]system-alert-00026: IPSec tunnel on int ethernet1 with tunnel ID 0x8
received a packet with a bad SPI. 217.0.72.117->193.108.181.253/56, ESP, SPI 0x0, SEQ 0x45080218
I 08/18/06 15:55:04 ports: port F1 is now off-line
I 08/18/06 15:55:06 ports: port F1 is now on-line
I 08/18/06 15:57:00 snmp: updated time by 4 seconds
2006-08-18 18:04:53 [Root]system-critical-00436: Large ICMP packet!
From 210.51.16.51 to 193.108.181.6, proto 1 (zone Untrust int ethernet1). Occurred 1 times.
2006-08-18 18:05:33 [Root]system-critical-00436: Large ICMP packet!
From 210.51.16.51 to 193.108.181.6, proto 1 (zone Untrust int ethernet1). Occurred 1 times.
I 08/18/06 19:15:24 ports: port F1 is now off-line
I 08/18/06 19:15:26 ports: port F1 is now on-line
2006-08-18 18:34:09 [Root]system-critical-00438: FIN but no ACK bit!
From 83.76.204.46:56242 to 193.108.181.101:6346, proto TCP (zone Untrust int ethernet1).
Occurred 2 times.
```

4. SLAD Nessus integration

As Nessus is a remote system integrity and vulnerability scanner it only provides the infrastructure. The communication between the scanning system with the Nessus server and the locally installed SLAD component is done via the SSH remote login protocol.



SLAD uses three NASL scripts to integrate into Nessus.

4.1. `slad_init.nasl`

The `slad_init.nasl` script takes the login parameter for the remote host. The username is hard-coded to "slad". This script takes the remote user's password or a public and private keypair for the login and stores these into Nessus' knowledge base. This script is in the ACT_INIT category, making sure it is run before the other scripts that rely on this information.

4.2. *slad_run.nasl*

The *slad_run.nasl* script presents the user with the options of which local scans to run, and actually connects to the remote system to start the jobs.

4.3. *slad_fetch_results.nasl*

The purpose of the *slad_fetch_results.nasl* script is to display the result of priorly submitted jobs. This script logs into the remote host and fetches a list of finished jobs from SLAD. These job's output then is retrieved and displayed to the nessus user.

4.4. Additionally used includes

Both the *slad_run.nasl* and the *slad_fetch_results.nasl* depend on two common included files. The first one, *slad_ssh.inc*, simplifies the GPL'ed Nessus SSH function script to use the knowledge base entries written by the *slad_init.nasl* script. The second one, *slad.inc*, actually defines the plugins that are available. This file is created from SLAD's plugin list with the *slad-build.awk* script which is part of the source distribution.

5. Expanding SLAD

All of SLAD's behaviour with regards to plugins is controlled by the *plugin.xml* in the plugin's directory. The *sladd* traverses its plugins directory on startup (which defaults to */opt/sladd/plugins*, but can be changed as described in section 2.5), looking for subdirectories that contain a *plugin.xml*.

5.1. General format of a plugin description

A plugin is partitioned into sets of programs that are to be run. Each plugin therefor is required to have at least one set, which in turn needs to have at least one entry.

The idea behind this to be able to present a user which radiobuttons for each plugin and let him select the one that fits him best, e.g. the desired scanning intensity. As a plugin might want to run multiple programs for a scan that might also differ between the sets, each program that needs to be run is declared as a separate entry within a set.

5.2. The *plugin.xml* format

The following shows a principle *plugin.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "plugin.dtd">

<slad:plugin xmlns:slad="http://slad.dnsystems.org/sladd" id="pluginid">
  <slad:pluginDescription>Plugin Description</slad:pluginDescription>
  <slad:pluginSet id="pluginsetid">
    <slad:pluginSetDescription>Set Description</slad:pluginSetDescription>
    <slad:pluginSetEntry id="entryid">
      <slad:pluginSetEntryBinary>bin/plugin</slad:pluginSetEntryBinary>
      [ <slad:pluginSetEntryCommandline>-option</slad:pluginSetEntryCommandline> ]
      [ <slad:pluginSetEntryRunAsUser>user</slad:pluginSetEntryRunAsUser> ]
      [ <slad:pluginSetEntryRunAsGroup>group</slad:pluginSetEntryRunAsGroup> ]
    </slad:pluginSetEntry>
    [ <slad:pluginSetEntry id="entryid"> ... </slad:pluginSetEntry> ... ]
  </slad:pluginSet>
  [ <slad:pluginSet id="setid"> ... </slad:pluginSet> ... ]
</slad:plugin>
[ <slad:plugin id="pluginid"> ... </slad:plugin> ... ]
```

Each *plugin.xml* file needs to have the usual XML header as shown above.

The start of a plugin declaration is the `slad:plugin` tag, where the plugin's id is set. The next step is the `slad:pluginDescription` tag which declares the text that will be shown

to the user on the plugin enable or disable level. An optional tag, `slad:pluginPath` may be given if the plugin's data and executable files are located outside the directory where the *plugin.xml* resides. Otherwise the plugin's path defaults to this directory.

Next a set is opened with the `slad:pluginSet` tag. Again, an id is given which must be unique among the sets of this plugin. Each set should have a description with the `slad:pluginSetDescription` tag unless there is only one set in the plugin. If no description is given, it is inherited from the plugin's main description.

Inside the plugin set the programs that have to be executed are declared within the `slad:pluginSetEntry` tag. Each entry within a set needs its own id. Each entry may have a `slad:pluginSetEntryDescription` tag, but this is not necessary as the current scripts do not present the entries to a user. The only mandatory tag inside an entry is `slad:pluginSetEntryBinary`, which names the binary to be run for this entry. The given executable and its path are interpreted relative to the plugin's path. If the executable requires command line arguments these are given with the optional `slad:pluginSetEntryCommandline` tag. Also optional are the tags `slad:pluginSetEntryRunAsUser` and `slad:pluginSetEntryRunAsGroup` that make the daemon change the user or group id before actually a program, potentially increasing security for programs that do not need root access.

5.3. A sample *plugin.xml*

The following shows a sample *plugin.xml* from the SLAD plugin "John the Ripper".

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "plugin.dtd">

<slad:plugin xmlns:slad="http://slad.dnsystems.org/slاد" id="john">
  <slad:pluginDescription>Execute John-the-Ripper to find weak
user passwords:</slad:pluginDescription>

  <slad:pluginSet id="fastjohn">
    <slad:pluginSetDescription>Fast-Crack: tries only usernames
and variations of these</slad:pluginSetDescription>
    <slad:pluginSetEntry id="fastjohn">
      <slad:pluginSetEntryBinary>bin/runjohn.sh
</slad:pluginSetEntryBinary>
      <slad:pluginSetEntryCommandline>--fast
</slad:pluginSetEntryCommandline>
    </slad:pluginSetEntry>
  </slad:pluginSet>

  <slad:pluginSet id="dictjohn">
    <slad:pluginSetDescription>Dictionary Mode (slow): tries all
words from a dictionary</slad:pluginSetDescription>
```

```
<slad:pluginSetEntry id="dictjohn">
  <slad:pluginSetEntryBinary>bin/runjohn.sh
</slad:pluginSetEntryBinary>
  <slad:pluginSetEntryCommandline>--dict
</slad:pluginSetEntryCommandline>
</slad:pluginSetEntry>
</slad:pluginSet>

<slad:pluginSet id="fulljohn">
  <slad:pluginSetDescription>Full-Crack (very slow): tries
dictionary and variations</slad:pluginSetDescription>
  <slad:pluginSetEntry id="fulljohn">
    <slad:pluginSetEntryBinary>bin/runjohn.sh
  </slad:pluginSetEntryBinary>
    <slad:pluginSetEntryCommandline>--full
  </slad:pluginSetEntryCommandline>
  </slad:pluginSetEntry>
</slad:pluginSet>

</slad:plugin>
```

5.4. Verifying the *plugin.xml*

After the new plugin has been installed, the correctness of the *plugin.xml* can be verified using the `sladd -s plugins` command. For the sample above it should show

```
p:john:checks for crackable user passwords using John
s:0:john:fastjohn:fast crack mode
e:0:john:fastjohn:fastjohn:fast crack mode
s:0:john:dictjohn:slow dictionary mode
e:0:john:dictjohn:dictjohn:slow dictionary mode
s:0:john:fulljohn:ultra slow full mode
e:0:john:fulljohn:fulljohn:ultra slow full mode
```

The listing shows the plugin (p), the three sets (s), and their entries (e).

5.5. Using the new plugin with Nessus

After the correctness has been verified, the *sladbuild.awk* script must be used to create a new *slad.inc* for Nessus. The full command would be

```
/opt/slad/bin/sladd -s plugins | awk -f sladbuild.awk >
  /usr/lib/nessus/plugins/slad.inc
```

with all paths showing the default install locations. This script creates a NASL fragment with four functions. The first function is used by the *slad_run.nasl* script to create the preferences page. The second function is for the same script and actually runs the plugins that have been selected. The third function is used by the *slad_fetch_results.nasl* script to display the descriptions for the plugins that are ready to present their output. The last function is for debug purposes only and can be used to display all plugins that are known to Nessus.

Note that for better fitting into Nessus the sets are named the “scan level” of the plugin, and entries are completely hidden from the user.

5.6. Golden rule - don't use dynamically linked files

SLAD should be able to act as a system and distribution independent software package. If you link files against libraries from a specific host system, some functions might not work on other hosts. Also with dynamically linked binaries called from SLAD there is always the potential risk of some trojan code manipulating SLAD output.

5.7. Golden rule - separate audit tools from the system

Try to put every directory having to do with SLAD under the `/opt/slاد` hierarchy. This will also make it easy to add and remove new binaries and dependent files.

A. John the Ripper

John the Ripper is a UNIX password cracker, currently available for UNIX (tested with Linux x86, FreeBSD x86, Solaris 2.x SPARC, OSF/1 Alpha), DOS, WinNT/Win95.

John the Ripper is designed to be both powerful and fast. It combines several cracking modes in one program, and is fully configurable for your particular needs (you can even code a custom cracking mode using the built in C compiler). Also, John is available for several different platforms, which enables you to use the same cracker everywhere (for example even continue a cracking session that you started on another platform).

John's crypt() routine is highly optimized for faster operation, which makes John run much faster than other crackers. This applies to both the assembly versions, and the portable pure C one.

John the Ripper supports the following cracking modes:

- wordlist with or without rules;
- "single crack", makes use of the login/GECOS information;
- incremental, tries all character combinations;
- external, allows you to define your own cracking mode.

A.1. Modification done by BOSS Team to John the Ripper

The "normal" version of John exposes cracked passwords in clear-text. This makes John difficult to operate with in a professional environment. Therefore, SLAD uses a John the Ripper version which has been patched for BOSS. In this version cracked passwords are not exposed anymore, instead only the user-accounts with crackable passwords are identified.

A.2. Configure John the Ripper's wordlists

The John wordlist used by SLAD can be found at `/opt/slاد/plugins/john/share/password.lst` in the default installation.

In a standard SLAD installation John is using a combined wordlist with dictionaries from the BSI (German Government) and the OpenWall Free Dictionary. This ensures that German and English words as well simple passwords are covered. For other languages you are required to extend this wordlist accordingly. To reduce John's processing time, it is recommend to re-sort and eliminate all non-unique words by the following commands:

```
#cat bydict >> new-wordlist
#cat wordlist >> new-wordlist
#sort -u new-wordlist > wordlist
```

A.3. Configuring John the Ripper's rule file

To do variations of the words in the wordlist, John uses a rule-file. This file defines the alterations and password variants of any john wordlist line.

The rule file can be found at */opt/slack/plugins/john/etc/john.ini*.

A.4. Ordinary commands

```
:      no-op - do nothing to the input word
<      n characters long, where n = 0-9
>n     reject word unless it is > n characters long, where n = 0-9
x      prepend character 'x' to word
$y     append character 'y' to word
l      force word to be lowercase
u      force word to be uppercase
c      force word to be capitalized
r      reverse word: "Fred" -> "derF"
d      duplicate word: "Fred" -> "FredFred"
f      reflect word: "Fred" -> "FredderF"
p      make best attempt to pluralize a lowercase word
onx    overstrike character in position n (start at 0) with character 'x'
inx    insert character 'x' in position n (start at 0) and shift the rest
        of the input string right
        nb: if n > strlen(input), character 'x' will be appended
xnm    extract substring from position n (start at 0) for up to m characters
```

A.5. Commands which may use character classes

```
sxy    replace (swap) all characters 'x' in the word with 'y'
s?cy   replace all characters of class 'c' in the word with 'y'
@x     purge all characters 'x' from the word
@?c    purge all characters of class 'c' from the word
!y     reject word if it contains character 'y'
!?c    reject word if it contains a character in class 'c'
/x     reject word unless it contains character 'x'
/?c    reject word unless it contains a character in class 'c'
=nx    reject word unless character at position n is equal to 'x'
=n?c   reject word unless character at position n is in class 'c'
```

nb: the word always starts at position 0

A.6. Character classes for use in the above commands

- ?? matches '?'
- ?v matches vowels: "aeiouAEIOU"
- ?c matches consonants: "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTUVWXYZ"
- ?w matches whitespace: " "
- ?p matches punctuation: ".,:;'?_!"
- ?s matches symbols
- ?l matches lowercase letters ('a' to 'z')
- ?u matches uppercase letters ('A' to 'Z')
- ?d matches digits ('0' to '9')
- ?a matches letters ('a' to 'z' and 'A' to 'Z')
- ?x matches letters and digits ('a' to 'z', 'A' to 'Z' and '0' to '9')

The complement of a class may be matched by the uppercase of its letter, i.e.: where ?d == DIGITS, ?D == NON-DIGITS, and so on.

A.7. New John commands

All the commands described above are the same as in Crack v4.1, while the following ones are added in John:

- { shift word left: "jsmith" -> "smithj", etc
- } shift word right: "smithj" -> "jsmith", etc
- Dn delete character in position n (start at 0) and shift the rest of the input string left
- P "crack" -> "cracked", etc (lowercase only)
- G "crack" -> "cracking", etc (lowercase only)
- i invert case, by keyboard: "Crack96" -> "cRACK(^\", etc
- I invert case: "Crack96" -> "cRACK96", etc
- v lowercase vowels: "Crack96" -> "CRaCK96", etc
- > shift each character right, by keyboard: "Crack96" -> "Vtsvl07",
- > etc
- < shift each character left, by keyboard: "Crack96" -> "Xeaxj85", etc

Extra "single crack" mode commands for word pairs support, to control if other commands are applied to the first, second, or both words:

- 1 first word only
- 2 second word only
- + the concatenation of both (should only be used after a '1' or '2')

If you use some of the above commands in a rule, it will only process word pairs (full names, from the GECOS information), and reject single words. A '+' is assumed at the end of any rule that uses some of these commands, unless you specify it manually. For example, '1l2u' will convert the first word to lowercase, the second one to uppercase, and use the concatenation of both. The use for a '+' might be to apply some more commands: '1l2u+r' will reverse the concatenation of both words, after applying some commands to them separately.

If a rule (line of commands) doesn't change a word, that word is rejected, unless the entire rule was a plain ':'. This assumes you have a plain ':' somewhere in your rules list.

The preprocessor is used to combine similar rules into one source line. For example, if you need to make John try lowercased words with digits appended, you could write a rule for each digit, 10 rules total. Now imagine appending two-digit numbers – the configuration file would get large and ugly.

With the preprocessor you can do these things easier. Simply write one source line containing the common part of these rules, and the list of characters you would have put into separate rules, in brackets (the way you would do in a regular expression). The preprocessor will then generate the rules for you (at John's startup). For the examples above, the source lines will be `'l$[0-9]'` (lowercase and append a digit) and `'l$[0-9]$[0-9]'` (lowercase and append two digits). These source lines will be expanded to 10 and 100 rules respectively. By the way, the preprocessors commands are processed right-to-left, and the characters are processed left-to-right, which gets a normal order of numbers in such cases as in the example with appending two-digit numbers. Note that I only used character ranges in these examples, however you can combine them with character lists, like `'[aeiou]'` will use vowels, and `'[aeiou0-9]'` will use vowels and digits.

There're some control characters in rules (`'['` starts a preprocessors character list, `'-'` marks a range inside the list, etc). You should prefix them with a `'` if you want to put them inside a rule without using their special meaning. Of course, the same applies to `'` itself. Also, if you need to start a preprocessor's character list at the very beginning of a line, you'll have to prefix it with a `':'`, or it would be treated as a new section start.

B. Tripwire rules

This version of SLAD comes with a generic Debian rule set for tripwire. For better results or to run the integrity services on another Linux-distribution you need to modify the tripwire rules.

The clear-text configuration-file is located at */opt/slad/plugins/tripwire/etc/twpol.txt*. After editing this file you need to compile and encrypt the local policy file. You can do this with the following command:

```
# twadmin -create-polfile -S site.key /opt/slad/tripwire/twpol.txt
```

Now, with the new tripwire policy you need to re-create the integrity database which can be done with:

```
# tripwire -init
```

C. Links and further information

- Checkrootkit Homepage <http://www.chkrootkit.org>
- Tripwire OpenSource Version <http://www.tripwire.org>
- ClamAV Homepage <http://www.clamav.net>
- John the Ripper Homepage <http://www.openwall.com/john/>
- Tiger Homepage <http://savannah.nongnu.org/projects/tiger>
- SLAD Homepage <http://www.dn-systems.org>
- Snort <http://www.snort.org>
- Logwatch and Trapwatch <http://www.logwatch.org>
- HOWTO-Customize-LogWatch
<http://www2.logwatch.org:8080/tabs/docs/HOWTO-Customize-LogWatch.html>