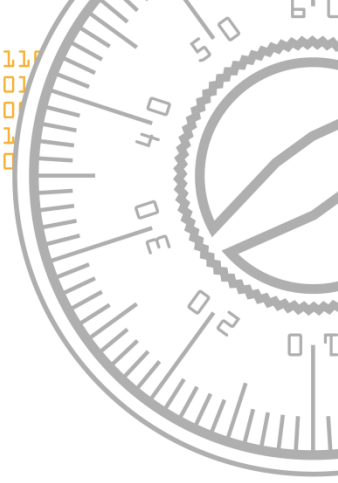


01-TSOLKAS-EXECUTIVE-CONSULTING-00100-ALEXANDER-TSOLKAS-001110001100100010100111
10110-AM-GROSSEN-STUECK-7-0111010101000-TEL+49-6158-747292-101001100010100101101
01010101011-64560-RIEDSTADT-01001-FAX+49-6158-747296-01010100011110001110010010
01010101-ALEXANDER@TSOLKAS.COM-0010010011-GSM+49-1577-1440251-01000101011011101
0010-WWW.TSOLKAS.COM-1010100100111-STID-DE254663527-11100000001111101001110110



HSS

High Security Server

Übersicht und Whitepaper

01010-POSTBANK-DUESSELDORF-10010101010101001-BLZ-44010046-01010101010101010110110010010100100011
101010101010010011-KONTO-998991467-0100111010101011000111011010100010101010010101001001110100100



Inhaltsverzeichnis

1.	Kurzübersicht.....	5
1.1.	Globale Einstellungen	5
1.2.	Capabilities	5
1.3.	Integrität	5
1.4.	Alleinstellungsmerkmal	6
2.	H S S Whitepaper.....	7
2.1	Gründe für den H S S	7
2.1	H S S Merkmale.....	8
3.	Analyse der Sicherheit auf Linux Systemen.....	8
3.1.	Benutzerbestimmte Zugriffskontrolle (DAC).....	8
3.2.	Debugging.....	8
3.3.	Capabilities	9
3.4.	Kernel-Module.....	9
3.5.	RAW-I/O.....	9
3.6.	Integrität.....	9
3.7.	Systemzeit.....	10
3.8.	Anforderungen an eine Lösung	10
4.	Alternativen.....	10
4.1.	RSBAC	10
4.2.	GRSecurity	10
4.3.	SELinux.....	11
4.4.	LIDS	11
4.5.	Sysrtrace.....	11
4.6.	Apparmor.....	11
4.7.	Gegenüberstellung der Sicherheit und Handhabung	12
4.8.	Vergleich	12
4.9.	Die Sicherheitsfunktionen	13
5.	Installationsanleitung	14
5.1.	Installation des Kernels (Debian 3.1)	14
5.2.	Installation der Dienstprogramme (Debian 3.1).....	14
5.3.	Installation des Kernels (generisches Linux)	15
5.4.	Installation der Dienstprogramme (generisches Linux).....	15
5.5.	Sonderfall: Installation von DRBD (= Digital Redundant Block Device).....	15
5.6.	Kernelkonfiguration	15
5.7.	Modulparameter	16
6.	Funktionen und Bedienung	16
5.8.	Die Verwaltungsanwendungen	16



5.9.	Dateien signieren.....	18
5.10.	Zusatz: Linux-Capabilities	18
5.11.	Sicherheitsattribut.....	19
5.12.	Das MD5 Sicherheitsattribut	20
5.13.	Das CAP Sicherheitsattribut.....	20
7.	Whitelist	22
8.	Verbosity	23

Bei Fragen zum H S S wenden Sie sich bitte an:

Alexander Tsolkas
Am großen Stück 7
64560 Riedstadt
Tel.: +(49)0 6158 747292
Fax: +(49)0 6158 747296
Gsm: +(49) 0 1577 1440251

HSS
High Security Server



Ein Hochsicherheitsserver
auf Linux-Basis

Open Source



1. Kurzübersicht

1.1. Globale Einstellungen

1. Versiegeln des H S S. Keine Module („Treiber“) können mehr geladen oder entladen werden. Der H S S kann nur durch Eingabe einer Passphrase und optional durch das Anschließen eines USB-Schlüssels entsperrt werden.
2. Es kann dem Administrator verboten werden die Zeit zurückzustellen.
3. Der Zugriff auf Prozessinformationen kann so eingeschränkt werden, dass Benutzer nur ihre eigenen Prozesse sehen.
4. Wie 1. nur ohne Versiegeln des H S S, d.h. Der H S S kann nur durch Eingabe einer Passphrase und optional durch das Anschließen eines USB-Schlüssels entsperrt werden.
5. Direktes Schreiben auf Block-Geräte kann verboten werden (dazu gehört z.B. MBR schreiben, Platten formatieren, Datenmanipulation auf Bit-Ebene, etc.).
6. Abschwächung von 5. Das direkte Schreiben auf Block-Geräte kann verboten werden, die bereits in das System eingebunden sind.
 - a. Als unveränderlich markierte Dateien¹ und Verzeichnisse können nicht mit dem *mount*-Befehl manipuliert werden.
7. Die Ausführung von Dateien ohne Hash kann verboten werden (Standard ist, die Ausführung von Dateien zu verbieten, wenn der Hash falsch ist).
8. Das Laden von Bibliotheken mit LD_PRELOAD² kann verhindert werden.
9. Es kann verhindert werden, dass SUID/SGID³ Flags gesetzt werden.
10. PTRACE kann verboten werden.
11. SHA1 kann anstatt MD5 als Hash verwendet werden.
12. Es kann eingestellt werden, das Hashs von Programmen und Dateien nur von einem Security Officer (SECOFF) verändert werden dürfen.

1.2. Capabilities

Der Zugriff auf die verschiedenen Funktionen des Kernels, ist geregelt über 31 „Fähigkeiten“ (den sogenannten Capabilities). Man muss eine bestimmte Fähigkeit besitzen, um eine bestimmte Kernel-Funktion zu nutzen. Bei einem Standard-Linux sind diese sehr einseitig verteilt. So hat der ROOT-Benutzer alle, ein normaler Benutzer hat fast keine. Der H S S hebt diese Beschränkung auf. Fähigkeiten lassen sich nun an beliebige Benutzer vergeben, dem Administrator können Fähigkeiten entzogen werden und es lassen sich Fähigkeiten an einzelne Programme binden. Letzteres ist eine elegante Methode SUID/SGID (s.o.) Flags zu ersetzen, denn es minimiert erstens die Angriffsfläche und ermöglicht zweitens ein Whitelisting auf dieser Ebene. Die Capabilities sind hier nicht einzeln aufgelistet. Genauere Beschreibung der 31 Capabilities gibt es hier: <http://linux.die.net/man/7/capabilities>

1.3. Integrität

¹ Das IMMUTABLE Flag ist ein Dateisystem-Attribut, dass auf Linux-Betriebssystemen gesetzt werden kann.

² LD_PRELOAD ist eine Umgebungsvariable, mit der es möglich ist vor Ausführung eine beliebige Bibliothek an ein Programm zu binden und Funktionen vorhandener Bibliotheken zu überschreiben.

³ SUID/SGID sind spezielle Rechte für ausführbare Dateien. Ein Programm mit SUID Recht wird im Kontext des Eigentümers des Programms ausgeführt, und nicht im Kontext des Benutzers. Da der Eigentümer meistens der Administrator ist, bergen solche Programme ein erhebliches Risiko.

Der Kernel-interne Integritätsschutz prüft Programme, Bibliotheken und direkt ausgeführte Skripte vor Ausführung gegen einen in den Metadaten abgelegten Hash (die Whitelist).

Zusätzlich ist eine Bibliothek vorhanden, die die Integrität beliebiger Dateien vor dem Öffnen prüfen kann. Zusätzlich kann die Bibliothek sogenannte Keyed-Hash Message Authentication Codes (HMAC) prüfen. Dabei wird beim Berechnen des Hashs ein geheimer Schlüssel angehängt. Eine Datei wird dann nur noch erfolgreich geprüft, wenn der Hash stimmt und der Signierende den geheimen Schlüssel kennt.

Zusätzlich gibt es ein Proof-of-Concept für ein Integritätsprüfungsmodul für den Apache-Webserver.

1.4. Alleinstellungsmerkmal

Bisher ist der H S S die einzige bekannte Sicherheitslösung auf Linux-Basis, die einen Integritätsschutz bietet. Dies verhindert eine persistente Kontamination des Systems mit Viren und Trojanern. Programme wie Virens Scanner oder Tripwire werden unnötig.

Andere Hersteller versuchen sich 4 Jahre nach dem ersten H S S mittlerweile auch auf diesem Gebiet. Zu nennen sind Microsoft und Apple. Anders als beim H S S ist die Integritätsprüfung auf den Betriebssystemen jedoch nicht „mandatory“ (systemgesteuert, verpflichtend), und ist daher als Schutz des Anwenders vor Viren und Trojaner völlig wirkungslos.

Die einzig bekannte verpflichtende Integritätsprüfung neben dem H S S findet auf dem iPhone von Apple statt, hier kann man gut erkennen, welche Hürden gegenüber Hackern und Crackern derartige Systeme bieten, selbst wenn man das Gerät in der Hand hält. Es wird aber auch deutlich, dass ein großer Hersteller erkannt hat, wie wichtig es ist gegen Viren und Trojaner sinnvolle Maßnahmen zu ergreifen. Viren und Trojaner auf einem Mobiltelefon können den Besitzer eine Menge Geld kosten.

2. H S S Whitepaper

2.1 Gründe für den H S S

Die Nachteile und Schwächen in Punkto Sicherheit aller herkömmlichen Betriebssysteme beruht auf deren Starrheit und Technik im Umgang mit Programmen und Skripten. Dies betrifft im besonderen Maße die folgenden Punkte:

- Virenerkennungssoftware funktioniert nur für bekannte Viren. Spezielle Makroviren und andere werden teilweise anhand bestimmter, bekannter Funktionen erkannt. Diese Erkennung bietet keine absolute Sicherheit, da sie reaktiv ist, und mit neuen Angriffsszenarien für die sie nicht programmiert sind, nicht fertig werden.
- Spähprogramme bzw. Trojanische Pferde können teilweise gar nicht erkannt werden. Das Einschleusen solcher Programme ist selbst von herkömmlichen Sicherheitseinrichtungen wie Firewalls und Virenerkennungsprogrammen nicht auszuschließen. Besonders hervorzuheben ist die Möglichkeit, Programmteile in dynamische Bibliotheken einzuschleusen, wodurch schädliche Funktionen durch jedes Programm ausgelöst werden kann. Werden Teile des Betriebssystems selbst infiziert, können alle herkömmlichen Sicherheitstechniken unterlaufen werden.
- Der allmächtige Root-Benutzer und alle Dienste, die in diesem Kontext gestartet werden, sind mit zu vielen Rechten ausgestattet, die sie nicht benötigen.
- Programmierfehler finden sich in jedem Programm. Leider haben Anwendungsprogrammierer selten die Qualifikation und das Problembewusstsein bzw. die Zeit, um die von ihnen erstellte Software nach Sicherheitsaspekten genau zu untersuchen. Seit je her besteht in der Unix- (und auch Windows-) Welt - ausgelöst durch die dort verbreitete de-Facto Standard Programmiersprache C das Problem des Buffer-Overflows. Im Wesentlichen werden dabei normale Programmfunktionen mit Eingaben außerhalb der gültigen Parameter angesteuert, was bei Programmen, die nicht nach dem Sicherheitsaspekt programmiert wurden zu unerwünschtem Verhalten führen kann.

Die traditionellen Angriffe auf Computersysteme beziehen sich auf zwei bevorzugte Techniken:

- Fehler in Programmen ausnutzen, um sie zum Absturz zu bringen, und auf der Betriebssystemebene schädliche Funktionen auszuführen.
- Passwörter herausfinden
- Die herkömmlichen Sicherheitssysteme auf Servern sind aufwendig in der Administration und eingeschränkt in der Wartbarkeit, Bedienbarkeit und Leistung.
- Bei geschlossenen Betriebssystemen, wie z.B. Microsoft, Novell etc. kann nicht herausgefunden werden, welche Funktionen oder Spionageteile diese schon ab Werk eingebaut haben.

Natürlich sollte man Sicherheitsprobleme an der Wurzel packen und nicht nur die Symptome bekämpfen, jedoch ist das nicht ohne weiteres möglich bei der Vielzahl der über Jahre entwickelten, komplexen Software. Beim H S S wird die Serversoftware wenn

nichts anderes gewünscht, immer vorrangig nach dem Sicherheitsaspekt ausgewählt. Wo dies nicht möglich erscheint, helfen die H S S -Sicherheitsfeatures.

Der Hochsicherheitsserver (H S S) ist eine Server Suite mit aktuellen Paketen der gebräuchlichsten Server-Programmen basierend auf Debian/GNU/Linux einer lizenzfreien Open-Source Betriebssystemdistribution mit einem Linux Kernel. Des Weiteren bietet der H S S spezielle Sicherheitsfeatures auf Betriebssystemebene, um ein System gegen Angriffe von innen und außen durch Hacker, Viren, etc. zu schützen. Die über das Maß der Open Source Distribution hinausgehenden Sicherheitsfeatures sind nur über die Lizenzsoftware (Control Panel) anzusteuern. Weiterhin unterstützt werden die Distributionen von SUSE und REDHAT.

2.1 H S S Merkmale

- Wirksamer ereignisgesteuerter Integritätsschutz mit MD5 Signaturen
- POSIX 1003.1e/1003.2c Access Control Lists
[\[1\] \(http://www.suse.de/~agruen/acl/linux-acls/online/\)](http://www.suse.de/~agruen/acl/linux-acls/online/)
- POSIX 1003.1e Capabilities mit Steuerung auf Dateibasis mittels Sicherheitsattributen
- Schutz des Speichers und der Modulschnittstellen des Linux-Kernels
- Schutz der Sicherheitsmerkmale durch Hardware-Token

3. Analyse der Sicherheit auf Linux Systemen

3.1. Benutzerbestimmte Zugriffskontrolle (DAC)

Bei der Discretionary Access Control oder benutzerbestimmten Zugriffskontrolle bestimmt der Eigentümer eines Objekts (einer Datei) selbst, wer die Objekte in welcher Weise nutzen darf. Diese Art der Rechteverwaltung ist fehleranfällig. Der Benutzer könnte z.B. die Objektrechte derart setzen, dass er selbst keinen Zugriff mehr hat, oder auf die Art und Weise, dass jeder Benutzer des Systems seine Objekte lesen oder schreiben kann.

Aus u.a. diesem Grund hat der Administrator auf derart verwalteten Systemen die Fähigkeit, die benutzerbestimmte Zugriffskontrolle zu überschreiben (CAP_DAC_OVERRIDE, CAP_DAC_READ_SEARCH), um auf diese Weise auf alle Objekte eines Systems zuzugreifen. Auf einem Linux-System hat auch jeder Prozess mit Root-Rechten diese Fähigkeit. Dies ist ein potentielles Sicherheitsrisiko. Ein derartiger Prozess, durch einen Buffer Overflow manipuliert, hätte Zugriff auf das gesamte System. Beim H S S lassen sich diese Privilegien mit dem CAP **Sicherheitsattribut** entziehen.

3.2. Debugging

Auf einem Produktionssystem ist Debugging nicht erwünscht, daher kann und sollte dies unterbunden werden. Eine beliebte Möglichkeit um Programme zu debuggen ist Ptrace [\[2\] \(http://www.die.net/doc/linux/man/man2/ptrace.2.html\)](http://www.die.net/doc/linux/man/man2/ptrace.2.html)). Der H S S verbietet grundsätzlich Ptrace auf den Init-Prozess. Die Ptrace-Fähigkeit kann Prozessen mit dem CAP **Sicherheitsattribut** entzogen werden.



3.3. Capabilities

Linux implementiert sog. Capabilities nach POSIX 1003.1e Standard um eine gewisse Kontrolle von Kernel-spezifischen Fähigkeiten über Subjekte zu haben (z.B. Net-Admin, DAC-Override, DAC-Read/Search). Jedoch ist die Vergabe dieser Capabilities in Linux etwas einseitig: Root-Prozesse haben alle Capabilities, normale Prozesse haben keine. Der H S S beseitigt dieses Problem, indem sich Root-Prozessen gesteuert Capabilities entziehen lassen. Siehe auch [Linux-Capabilities](#).

3.4. Kernel-Module

Die gefürchtete Linux-Kernel Backdoor. Der Linux Kernel ist monolithisch. Ein Kernelmodul hat, einmal geladen, die gleichen Privilegien wie der Kernel selbst. Daher bieten sich einer Backdoor auf Kernelebene alle Möglichkeiten sich nicht auffindbar zu machen, und alle Rechte auf dem System zu erhalten. Ein H S S Sicherheitsmechanismus verhindert, dass Module ab einem bestimmten Zeitpunkt geladen werden dürfen.

3.5. RAW-I/O

Bitweises Schreiben auf Blockgeräte (z.B. Festplatten), rohen Speicher, Kernelspeicher sind Dinge die ein Standard Linux-Kernel zulässt. Auf einem Produktivsystem ist RAW-I/O in der Regel nicht erwünscht, es sollte die API Abstraktionsschicht des Betriebssystems verwendet werden. Der H S S kann den Zugriff auf rohen Kernel-Speicher verbieten. Bei Festplatten bietet der H S S ein mehrstufiges Konzept an, bei dem zwischen "lebenden", also sich im Betrieb befindlichen Platten, und solchen die noch nicht in Betrieb genommen wurden, unterschieden wird.

3.6. Integrität

Die Integrität eines Systems ist gefährdet durch Daten, die von unberechtigten Personen auf das System gebracht werden. Insbesondere betrifft das Anwendungen und die Systemkonfiguration. Trojaner sind Anwendungen die in der Regel einen gewünschten Dienst anbieten, zusätzlich aber über verdeckte Funktionen verfügen. Eine Spezialform des Trojaner, die sogenannte Backdoor ist auf Servern am häufigsten anzutreffen. Ein Angreifer versucht dabei meist die Anwendung eines Systemdienstes zu ersetzen gegen eine Anwendung, die ebenfalls diese Dienste anbietet, jedoch zusätzlich dem Angreifer einen verlässlichen Zugang zum System verschafft.

Viren, selten ein Server-Problem, hängen ihren Programmcode an ein Wirtsprogramm, um Schaden zu verursachen und um sich zu verbreiten. Diese Probleme versucht man über integritätssichernde Maßnahmen zu bekämpfen.

Das Problem ist, dass integritätsprüfende Werkzeuge (z.B. Tripwire, Virens Scanner) reaktiv bzw. zeitgesteuert arbeiten und Verletzungen der Integrität in der Regel zu einem Zeitpunkt feststellen an dem der Schaden bereits da ist. Der H S S hat einen Integritätsschutz im Betriebssystem. Die Integritätsprüfung des Programmcodes einer Anwendung findet immer statt, kurz bevor sie ausgeführt wird. Schlägt diese Prüfung fehl, wird die Ausführung abgebrochen und gleichzeitig in den Systemlogs vermerkt. Der Effekt ist unmittelbar, der Schaden gering. Eine Backdoor kann im H S S nie geöffnet werden, ein Virus sich nie verbreiten oder Schaden anrichten.



3.7. Systemzeit

In einem verteiltem System ist Kausalität eine wichtige Voraussetzung (z.B. für Transaktionen) d.h. die Ursache muss immer vor der Wirkung kommen, die Zeitachse darf nur vorwärts laufen. Zentrale Zeitgeber z.B. der ntpd kennen diese Regeln. Wird festgestellt das eine Rechneruhr vorgeht, werden diese Dienste zur Wahrung der Kausalität niemals versuchen die Systemzeit "zurückzudrehen", die Systemzeit wird einfach so lange langsamer vergehen, bis die Systemuhr wieder mit dem zentralen Zeitgeber übereinstimmt. Das Manipulieren der Systemzeit ist jedoch gängige Praxis bei Angreifern um auf dem System z.B. Logs zu verschleiern. Der H S S verbietet das Zurücksetzen der Systemzeit.

3.8. Anforderungen an eine Lösung

- Es sollte ein Kompromiss aus leichter Wartbarkeit, wenigen Einschränkungen im Normalbetrieb auch für sicherheitsunbedarfte Administratoren, und guter Sicherheit sein.
- Es sollte durch speziell berechnigte Personen komplett zur Laufzeit hinzufügar und entfernenbar sein.
- Es sollte auf modernen Linux Systemen nachrüstbar sein.
- Es sollte zur Laufzeit wartungsarm sein und das Change-Management keinesfalls komplizieren, z.B. durch das Nachinstallieren von Diensten, das Anlegen neuer Benutzer etc.
- Es sollte möglichst viele Sicherheitsverbesserungen "out-of-the-box" d.h. ohne Anpassung des Systems hervorbringen.

4. Alternativen

Es gibt mehrere Ansätze und Lösungen, um Sicherheit auf Linux-System zu erzeugen. Hier eine Auflistung:

4.1. RSBAC

Vorteile:

- echte Ressourcenbeschränkung
- BSD Jails
- Bell-La Padula MAC Modell + Erweiterung dieses Modells um dessen Schwächen auszugleichen

Nachteile:

- komplexe Konfiguration ohne annähernd ausreichende Dokumentation
- erfordert Kernel-Patching, kein LSM

4.2. GRSecurity

Das Projekt GRSecurity von Brad Spengler erweitert die Kernel 2.4 und 2.6 ebenfalls um Sicherheitsfunktionen wie Mandatory Access Control und Schutz vor Buffer Overflows, integriert sich dabei allerdings nicht über LSM in das System. Stattdessen besteht GRSecurity aus einem ganzen Satz von Patches für den Kernel. Das dürfte ein Grund sein, warum keine der großen Linux-Distributionen dieses System standardmäßig einsetzt. Mit GRSecurity lassen sich detailliert die Rechte von Prozessen und Anwendern

auf Dateien und ihre Capabilities festlegen. Bei der Konfiguration wird der Anwender durch ein Lern-System unterstützt, welches zunächst nur das System beobachtet und daraus auf Wunsch eine Policy erstellt, und aktiviert. Das MAC/ACL-System ist derzeit nicht dokumentiert und daher praktisch nicht einsetzbar. Es besteht theoretisch die Möglichkeit die einzelnen Sicherheitsfeatures über das SYSCTL-Interface zu steuern, was aufgrund der teilweise stark funktionseinschränkenden Sicherheitsfeatures notwendig ist. Jedoch liegt auch hier der Nachteil, dass es aufwendig zu integrieren ist.

Vorteile:

- Lösung mit effektivem Buffer-/Heap-Overflow Schutz

Nachteile:

- Erfordert Kernel-Patching. Die Eingriffe in den Kernel sind dabei dermaßen massiv, das der Patch zu jeder Kernel-Version angepasst werden muss.
- Mit PAX Patches nur auf i386 Plattform lauffähig.
- Schlecht/nicht dokumentiertes MAC/ACL-System.
- MAC/ACL System schwer wartbar/administrierbar.
- Die anderen Sicherheitsfeatures schränken das System sehr ein.
- Unklare Zukunftsaussichten.
- Das Projekt hängt aufgrund der fehlenden LSM-Integration sehr hinterher. Für den 2.6 Kernel sind die interessanten Features noch nicht implementiert.

4.3. SELinux

Vorteile:

- Im offiziellen Kernel
- Breite Unterstützung
- Sponsor/Projektverantwortlicher ist die NSA
- Nutzt die offiziellen LSM-Schnittstellen

Nachteile:

- Extrem schwer zu warten/administrierbar
- Verführt Administratoren zur Nachlässigkeit wodurch der Schutz obsolet wird

4.4. LIDS

- Ist noch in etlichen Installationen im Einsatz, ist jedoch veraltet.

4.5. Systrace

- Biete nur schwachen Schutz, und ist sehr umständlich in der Handhabung.

4.6. Apparmor

- Biete nur mittleren Schutz, und ist umständlich in der Handhabung.

4.7. Gegenüberstellung der Sicherheit und Handhabung

	SICHERHEIT	HANDHABUNG
Apparmor	++	-
SELinux	+++	---
Lids	+	-
Systrace	+	-
RSBAC	+	--
GRSecurity	+	--
H S S	+++	++

4.8. Vergleich

Der H S S setzt wie SELinux und Apparmor auf das LSM-Konzept auf und geht damit den "offiziellen" Weg, wie Sicherheit im Linux-Kernel etabliert werden sollte. Der H S S hat damit die gleichen Vorteile gegenüber GRSecurity, Systrace und Lids.

Der Schutz von SELinux und Apparmor funktioniert nur dann, wenn das System entsprechend konfiguriert ist. Eine Lücke oder eine "Abkürzung" durch den Administrator machen den Schutz wirkungslos.

Der H S S bietet ein "Härtung" seines Kerns von seiner Aktivierung an, und bleibt dabei kompatibel zu allen Anwendungen. Ausnahme: Anwendungen, die mit Benutzerrechten versuchen auf "rohen" Speicher zuzugreifen (dazu gehören z.B. grafische Oberflächen wie z.B. X11). Die einzige alternative Sicherheitslösung, die ähnliches macht, ist GRSecurity.

SELinux und Apparmor bieten keinen grundlegenden Schutz gegen Schwächen des Linux-Kernels ein.

Der H S S hat ein einfaches globales Levelsystem. Anwendungen müssen nicht angepasst werden. Dafür ist er nicht so flexibel. Sowohl SELinux als auch Apparmor stellen Ansprüche an die Anwendungen. Bei Apparmor müssen alle Anwendungen, die von den Mechanismen profitieren wollen, angepasst sein. Das verursacht Integrationsprobleme mit proprietärer Software. Bei SELinux müssen bestimmte Verwaltungsdienste und -Programme auf die veränderte Sicherheitsumgebung angepasst werden. Diese sind in der Regel Open-Source-Lösungen. SELinux hat kaum Probleme mit proprietärer Software. Der H S S hat gar keine Probleme damit. Jede Software, ob proprietär oder nicht, kann signiert werden, weitere Anpassungen des Systems sind nicht notwendig.

SELinux ist sicherheitstechnisch sehr gut. Die Konfiguration ist leider sehr aufwendig, was der Grund dafür ist, dass man es in der Praxis nur selten sieht. Apparmor ist feiner einstellbar (mit der gleichzeitigen Notwendigkeit das auch zu tun). H S S und Apparmor setzen sicherheitstechnisch an der gleichen Stelle an (beide greifen bei den POSIX-

Capabilities ein). Der H S S hat jedoch noch mehr Features und Ansatzpunkte und beschränkt sich nicht nur auf POSIX-Capabilities. Die "Grundhärtung" des Systems ist beim H S S wirkungsvoller.

Ein H S S-Feature, das keine andere Sicherheitslösung bietet, ist der Malware-Schutz. Alle ausführbaren Dateien auf dem System können nur ausgeführt werden wenn sie eine gültige Signatur haben. Trojaner und Viren können das System nicht infizieren, da infizierte Programme nicht mehr ausgeführt werden. Das Betriebssystem hat eine eingebaute Integritätsprüfung, die immer greift, bevor Schaden entstehen kann.

Ein exklusives Feature von GRSecurity ist ein wirksamer Buffer-Overflow-Schutz. Das Problem ist, dass er nur auf Intel-Systemen funktioniert, und es ist definitiv nicht mit dem LSM-Konzept zu erfüllen.

Praxis: Um SELinux gut und sicher zu konfigurieren (in dem Fall würde es sicherheitstechnisch den H S S fast erreichen) braucht man pro System ca. 15 Manntage. GRSecurity ist derzeit auf keinem Stand, der produktiv einsetzbar ist; es hat auch nur wenig Dokumentation. Lids ist veraltet, wenn auch in gewissen Kreisen noch verbreitet; Systrace ist ein schwaches System.

4.9. Die Sicherheitsfunktionen

Herkömmliche H S S-Installationen mussten noch eine Einstellung des Sicherheitslevels haben. Das ist seit 8/2008 nicht mehr der Fall. Es gibt jetzt „Sicherheit á la carte“. In den meisten Fällen kann auf manuelle Eingriffe verzichtet werden. Der H S S stellt automatisch die beste Kombination ein!

Deshalb sind die nachstehenden Einstellungen eher als Hinweise zu betrachten:

- Ein Ptrace auf den Init-Prozess ist nicht erlaubt.
- Programme, die eine ungültige Signatur im **Sicherheitsattribut** haben, werden nicht ausgeführt.
- Audit des Systems: alle Verstöße gegen unerlaubte Attribute bzw. fehlende und falsche Checksummen der Whitelist werden geloggt.
- Audit des Systems: Anfragen und Ablehnungen von POSIX-Capabilities werden geloggt (siehe auch **Verbosity**).
- Das Laden/Entladen von Kernelmodulen ist nicht mehr möglich.
- Das Ändern von IMMUTABLE und APPEND Flags ist nicht mehr möglich.
- RAW-I/Os (z.B. Schreiben auf Kernelspeicher, etc.) sind nicht mehr möglich. Ab diesem Modus laufen z.B. grafische Oberflächen auch nicht mehr.
- Das direkte Schreiben auf gemountete Blockgeräte ist nicht mehr möglich (z.B. Bootloader manipulieren, Festplatte formatieren, Partitionen ändern).
- Das SETSUID/SETSGID setzen ist nicht mehr möglich.
- Die Manipulation der Systemzeit ist nicht möglich.
- Das Mounten auf IMMUTABLE Mountpoints ist nicht mehr möglich.
- Ausführbare Dateien die nicht eingefroren (IMMUTABLE) sind werden nur mit gültigem Hash (**Whitelist**-Verfahren) ausgeführt.
- Bei ausführbaren Dateien ohne Capability-Sicherheitsattribut werden die Capabilities gemäß **gcap** reduziert.

- Das direkte Schreiben auf **alle** Blockgeräte ist nicht mehr möglich (z.B. das Formatieren der Festplatte, Bootloader manipulation, Partitionen ändern).
- Das Unmounten von Mountpoints die als IMMUTABLE gekennzeichnet sind, ist nicht mehr möglich.
- Die Libc wird nur noch im sicheren Modus benutzt, Programmlader-Optionen wie z.B. LD_PRELOAD funktionieren nicht mehr.
- Netzadministration (Netzadressen ändern, etc.) nicht mehr möglich.
- Reboot nicht mehr möglich.
- Verändern der erweiterten Sicherheitsattribute nicht mehr möglich (z.B. Whitelist-Checksummen, ACLs).
- Dateirechte-Override des root-Benutzers nicht mehr möglich.
- Mounten/Unmounten von Dateisystemen nicht mehr möglich.
- RAW-Socket Netzwerkzugriff nicht mehr möglich.

5. Installationsanleitung

5.1. Installation des Kernels (Debian 3.1)

Für Debian existieren fertige Pakete (hss-modutils-2.6.x.x-hss) passend zu einer Kernel-Version (kernel-image-2.6.x.x-hss). Diese werden auf das betreffende System geladen und folgendermaßen installiert:

```
hss:~# dpkg -i kernel-image-2.6.17.6-hss_1_i386.deb
```

Wird der GRUB-Bootloader verwendet (Standard) sind keine weiteren Schritte zur Installation des H S S-fähigen Kernels nötig. Das System muss nun neu gestartet und der neue Kernel gebootet werden.

```
hss:~# reboot
```

Nach dem Neustart ist sicherzustellen, dass das System weiterhin einwandfrei funktioniert, und die Hardware erkannt wurde. Anschließend fährt man fort mit der Installation der Dienstprogramme für den H S S .

5.2. Installation der Dienstprogramme (Debian 3.1)

```
hss:~# dpkg -i hss-modutils-2.6.17.6-hss.deb
```

Die Installationsroutinen kopieren Start- und Verwaltungsskripte und verankern diese im Runlevel-Control-System des Debian, um die H S S-Sicherheitsweiterungen beim Systemstart zu aktivieren. Abschließend wird zur Eingabe einer Passphrase aufgefordert, mit dieser kann man die H S S -Sicherheitserweiterungen zur Laufzeit deaktivieren.

```
hss:~# hssconf.sh -load
```

aktiviert den H S S mit den Grundeinstellungen. Grundeinstellungen für Teststellungen sind derzeit:

- Sicherheitlevel 0
- Passphrase erlaubt Deaktivieren der Sicherheitseinstellungen
- Globale Capability Maske auf -1 (entspricht 0xffffffff, Capabilities werden nicht entzogen)



5.3. Installation des Kernels (generisches Linux)

Wird eine andere Distribution verwendet (z.B. SuSE, Redhat) kann das H S S-Kernelmodul für den Kernel aus den Quellen erzeugt werden. Der bestehenden Kernel kann weiter verwendet werden, wenn folgende Dinge zutreffen:

- Version > 2.6.10
- Alle unterstützten Dateisysteme unterstützen auch erweiterte Attribute, Sicherheitattribute und POSIX-ACLs
- POSIX-Capabilities wurden als Kernel-Modul eingebunden
- Kernel läuft auf dem System und die Kernel-Quellen sind vorhanden

Treffen einer oder mehrere Punkte nicht zu, muss in dem System ebenfalls der Kernel ersetzt werden. Es wird empfohlen sich beim Erzeugen eines neuen Kernel möglichst an die Standardkonfiguration der jeweiligen Distribution zu halten und nur die oben genannten Punkte anzupassen. Lesen Sie [Kernelkonfiguration](#)

5.4. Installation der Dienstprogramme (generisches Linux)

Entpacken des Archives

```
hss:~# tar xvfz hss-modutils-generic.tgz
```

Wechsel in das Archivverzeichnis und Aufrufen des Installationskripts

```
hss:~# cd hss-modutils-generic; ./install.sh
```

5.5. Sonderfall: Installation von DRBD (= Digital Redundant Block Device)

Zur Gewährleistung eines Hochverfügbarkeits-Servers. Zwei Server sind mit DRBD verbunden.

```
hss:~# dpkg -i drbd-2.6.17.6-hss.deb
```

5.6. Kernelkonfiguration

Die Kernelkonfiguration wird derart durchgeführt, dass die hier folgenden Variablen gemäß der Vorgabe angepasst werden.

Konfigurationsvariablen für den Kernel

```
CONFIG_*_FS_XATTR=y
```

```
CONFIG_*_FS_POSIX_ACL=y
```

```
CONFIG_*_FS_SECURITY=y
```

(* ist Platzhalter für den Namen des Dateisystems, z.B. REISERFS, EXT3, EXT2, JFS, XFS)

```
CONFIG_SECURITY=y
```

```
CONFIG_SECURITY_CAPABILITIES=m
```

```
CONFIG_SECURITY_SECLVL=m
```



5.7. Modulparameter

Der H S S kann bei Aktivierung mit verschiedenen Modulparametern gestartet werden. Die verwendeten Modulparameter stehen in der /etc/hss.conf, die Standardeinstellungen sind in aller Regel ausreichend.

NAME	TYP	GÜLTIGE WERTE	STANDARD
Gcap	uint	globale capabilities Bitmaske (s.gcap)	-
Verbosity	integer	0-3 (s. verbosity)	0
Use_hardware_token	boolean	0=kein hardware token erforderlich 1=hardware token muss da sein um den H S S zu entsperren	0
Vendor_id	uint	USB-Hersteller-ID des tokens	-
Produkt_id	uint	USB-Produkt-ID des tokens	-

6. Funktionen und Bedienung

5.8. Die Verwaltungsanwendungen

Bedienen der Sicherheitslevel (mit Management-Tool)

Mit dem Management-Tool lassen sich folgende Befehle ausführen:

```
# hssconf.sh
```

```
Usage: /usr/sbin/hssconf.sh {-newpass|/load|/lock|/unlock|/mark|/unmark}
```

-newpass:

Mit dieser Option lässt sich die H S S -Entsperrungs-Passphrase neu setzen. Dafür muss sich der H S S in Sicherheitslevel 0 befinden.

```
# hssconf.sh -newpass
```

NEW H S S PASSPHRASE:

RETYPE NEW PASSPHRASE:

-mark:

Dateien als IMMUTABLE zu markieren sorgt dafür, dass sie nicht, selbst nicht durch den Root-Benutzer, verändert werden können.

Mit dem Befehl `# hssconf.sh -mark` werden die wichtigsten Systembestandteile automatisch als IMMUTABLE markiert. Für IMMUTABLE Dateien führt der H S S die Integritätsprüfung nicht durch, was die Performance verbessert. Beachten Sie bitte: ein IMMUTABLE markiertes System kann in seinen wesentlichen Eigenschaften nicht verändert werden, und damit sind auch keine Updates von Programmen möglich. Um Änderungen an Modulen und den Sicherheitseinstellungen durchführen zu können muss man den H S S entsperren:

Achtung! Sie wollen die H S S Sicherheitsfunktionen deaktivieren!

H S S UNLOCK PASSWORD:

anschließend muss man die IMMUTABLE-Markierung aufheben mit:

`# hssconf.sh -unmark`

Man beachte, dass in der Standardeinstellung der H S S beim Übergang in den unsicheren Modus die primäre Netzwerkschnittstelle deaktiviert. Dies ist der Wartungsmodus: Für Internet-Rechner bei denen davon ausgegangen wird, dass die Primäre und eine sekundäre Netzwerkschnittstelle exklusiv für das interne Netz zur Verfügung steht.

Dateien als IMMUTABLE zu markieren funktioniert auch mit dem GNU Werkzeug `attr`. Siehe dazu die MAN-Page von `attr`:

`# man attr`

Bedienen der Sicherheitslevel (mit Skript-Schnittstelle)

Die H S S Sicherheitserweiterungen haben eine `sysfs`-Schnittstelle. Abgefragt und erhöht werden kann der Level durch lesen bzw. schreiben an `sysfs`.

H S S Sperren:

```
hss:~# echo 1 > /sys/kernel/security/hss/lock
```

Sperrstatus abfragen

```
hss:~# cat /sys/kernel/security/hss/lock
```

1

```
hss:~#
```

Entsperrt werden kann der H S S, wenn der Modulparameter `sha1_password` angegeben wurde, durch schreiben der Passphrase an das `sysfs`.

```
hss:~# echo "passphrase" > /sys/kernel/security/hss/passwd
```

Wurde `sha1_password` beim Laden des Sicherheitsmoduls nicht angegeben, kann der H S S nicht entsperret werden. Ein Neustart des Systems ist nötig.

5.9. Dateien signieren

Mit dem Befehl

```
# whitelist.sh md5 <dateiname>
```

wird von der benannten Datei eine MD5-Prüfsumme erzeugt und im H S S -System bekannt gemacht. Bei Ausführung des Befehls wird die MD5-Prüfsumme angezeigt. Sie sollte mit Herstellerangaben des Programms abgeglichen werden, wenn möglich. Dieses Kommando kann nur der Security Officer (SECOFF) anwenden.

Alle ausführbaren Dateien eines Systems signieren

Folgender Befehl findet alle ausführbaren Dateien im System, signiert diese und fügt sie der Whitelist hinzu.

```
find / -type f -perm -111 -exec whitelist.sh md5 {} \;
```

Alle Dateien mit SUID-Flag in Datei /root/suid.txt schreiben

Tipp: Besonders sensitive Programme sind Programme mit SUID-Bit gesetzt, diese finden Sie mit dem Befehl:

```
find / -type f -perm -4000 -fprintf /root/suid.txt '%#m %u %p\n'
```

5.10. Zusatz: Linux-Capabilities

Ein Prozess mit ROOT-Rechten hat in der Regel alle Capabilities bis auf CAP_SETPCAO, ein normaler Benutzer keine. Linux implementiert etwa 6 der im Posix 1003.1e Standard festgelegten Capabilities plus 24 eigene, sogenannte Linux-Capabilities. Da die Fähigkeiten, die ein Dienst besitzen muss, sehr unübersichtlich zu handhaben sind, bietet der H S S zwei Möglichkeiten, die Dienste zu verwalten:

- Alles ist erlaubt und Ausnahmen gelten als Restriktionen (einfach, wartungsarm)
- Alle ist verboten und Ausnahmen sind Erweiterungen (aufwendiger und sicherer)

Capability	Wertigkeit
CAP_CHOWN	0
CAP_DAC_OVERRIDE	1
CAP_DAC_READ_SEARCH	2
CAP_FOWNER	3
CAP_FSETID	4
CAP_KILL	5
CAP_SETGID	6
CAP_SETUID	7



Capability	Wertigkeit
CAP_SETPCAP	8
CAP_LINUX_IMMUTABLE	9
CAP_NET_BIND_SERVICE	10
CAP_NET_BROADCAST	11
CAP_NET_ADMIN	12
CAP_NET_RAW	13
CAP_IPC_LOCK	14
CAP_IPC_OWNER	15
CAP_SYS_MODULE	16
CAP_SYS_RAWIO	17
CAP_SYS_CHROOT	18
CAP_SYS_PTRACE	19
CAP_SYS_PACCT	20
CAP_SYS_ADMIN	21
CAP_SYS_BOOT	22
CAP_SYS_NICE	23
CAP_SYS_RESOURCE	24
CAP_SYS_TIME	25
CAP_SYS_TTY_CONFIG	26
CAP_MKNOD	27
CAP_LEASE	28
CAP_AUDIT_WRITE	29
CAP_AUDIT_CONTROL	30

5.11. Sicherheitsattribut

Sicherheitsattribute werden vom Dateisystem als Metainformation zu einer Datei gespeichert. Sie sind eine Spezialform der erweiterten Attribute. Erweiterte Attribute erlauben Namensräume für Metainformationen einer Datei. Sicherheitsattribute haben den Prefix security. Andere Prefixe sind für Linux-Dateisysteme derzeit nicht

implementiert, so dass nur erweiterte Attribute mit dem Prefix security mit den vorhandenen Userland Werkzeugen erzeugt werden können.

Der H S S führt zwei Sicherheitsattribute ein, eines um zu einer Datei einen MD5 Hashwert zu speichern und eines um eine Capabilities-Bitmaske zu einer ausführbaren Datei zu speichern.

5.12. Das MD5 Sicherheitsattribut

Dieses Sicherheitsattribut speichert den (Soll) MD5 Hashwert zu einer Datei in den Metadaten der Datei

```
hss:~# getfattr -n security.md5 <datei>
```

gesetzt werden kann es folgendermaßen: (hier am Beispiel der Datei /bin/cat)

```
hss:~# setfattr -n security.md5 -v 388b2a370b29026d36ba484649098827 /bin/cat
```

gelöscht werden kann es mit:

```
hss:~# setfattr -x security.md5 /bin/cat
```

5.13. Das CAP Sicherheitsattribut

Ein (relativ unpraktisches) Beispiel, das aber die Mächtigkeit dieses Sicherheitsattributes zeigt:

Der Root-Benutzer möchte den privaten Schlüssel eines Benutzers auslesen.

```
hss:~# /bin/cat /home/jrandomuser/.ssh/id_dsa
```

```
-----BEGIN DSA PRIVATE KEY-----
```

```
GEHEIMGEHEIMGEHEIMGEHEIM
```

```
-----END DSA PRIVATE KEY-----
```

Und kann das selbstverständlich durchführen.

Nun wird /bin/cat mit folgendem Sicherheitsattribut versehen:

```
hss:~# setfattr -n security.cap -v fffffef9 /bin/cat
```

Die Capability-Maske 0xffffef9 entzieht die Capabilities CAP_DAC_OVERRIDE, CAP_DAC_READ_SEARCH und CAP_SETPCAP.

Der Root-Benutzer versucht es nochmals.

```
hss:~# /bin/cat /home/jrandomuser/.ssh/id_dsa
```

```
/bin/cat: /home/jrandomuser/.ssh/id_dsa: Keine Berechtigung
```

Es funktioniert nicht, da das Programm /bin/cat nicht die CAP_DAC_OVERRIDE Fähigkeit nutzen darf.



Viele Administratoren oder Operatoren haben sich in ihrer Anfangszeit folgendes Szenario vorgestellt: Ein Benutzer hat Schreibrechte auf eine Datei die ihm gehört. Damit kann er die Rechte an seiner Datei ändern. Er löscht alle Rechte der Datei inklusive seiner eigenen Lese- und Schreibrechte. Was ist passiert? Er hat sich ausgesperrt. Wie löst man das Problem?

Die Capability `CAP_DAC_OVERRIDE` kommt ins Spiel. Der technische aber sprechende Name bedeutet die Fähigkeit (CAP - Capability) die benutzerbestimmte Zugriffsteuerung (DAC - Discretionary Access Control) zu überstimmen (OVERRIDE).

Diese Fähigkeit ist also durchaus wichtig für einen Administrator um falsch gesetzte Dateirechte zu reparieren. Es verleitet aber auch dazu die Datei-Rechtevergabe nicht richtig zu pflegen, denn falls irgendein Programm oder Dienst auf etwas nicht zugreifen kann, wird es im Root-Kontext gestartet, weil es damit aufgrund der Fähigkeit funktioniert, egal wie die Rechte gesetzt sind. Ein Dienst mit dieser Fähigkeit bedeutet aber die Gefahr von Informationssicherheitslücken. Nur ein eingeloggter Administrator oder ein Dienst, der die Aufgabe hat Dateirechte zu reparieren, sollten mit einer derartigen Fähigkeit ausgestattet sein.

Das CAP-Sicherheitsattribut ist besonders sinnvoll bei Programmen mit SUID-Flag. Das spezielle Flag sorgt dafür, dass das Programm nicht im Kontext des Benutzers sondern im Kontext des Dateieigentümers ausgeführt wird. Das SUID-Flag wird sicherheitstechnisch kontrovers diskutiert, da es ein Risiko darstellt, gleichzeitig einen großen Mehrwert bietet.

Ein (relativ nützliches) Beispiel: Das Programm `/bin/ping`. Die ausführbare Datei gehört dem Root-Benutzer und hat das SUID-Flag gesetzt. Dadurch kann jeder Benutzer das Programm nutzen als wäre dieser der Root-Benutzer, was auch nötig ist, da normale Benutzer nicht das ICMP-Protokoll nutzen dürfen (dazu wird `CAP_NET_RAW` benötigt).

Ping könnte z.B. ein CAP-Sicherheitsattribut erhalten das nur `CAP_NET_RAW` enthält. Die Bitmaske wäre in diesem Fall `0x00002000`

```
hss:~# setfattr -n security.cap -v 00002000 /bin/ping
```

Das Programm hat weiterhin seinen vollen Funktionsumfang. Es wurden nur die Fähigkeiten entzogen, die es für seine spezielle Aufgabe gar nicht benötigt. Ein Angreifer als nicht privilegierter Benutzer, der z.B. einen eventuellen Fehler in dem Programm nutzen möchte, um Root-Rechte zu erhalten, kann dies nicht: es bleibt ihm nur eine von 26 root Capabilities um Schaden anzurichten.

Sinnvolle Anwendungen findet man auch bei allen Diensten deren einziger Grund, mit Root-Rechten zu laufen ist, das sie an einen privilegierten Port (<1024) binden. Die Capability-Bitmaske die nur `CAP_NET_BIND_SERVICE` erlaubt ist `0x00000400`.

Folgenden Bildschirmausgabe zeigt wie man anzeigen kann mit welchen Capabilities ein Prozeß läuft

```
hss:~# cat /proc/3425/status
Name: inetd
State: S (sleeping)
SleepAVG: 29%
Tgid: 3425
Pid: 3425
PPid: 1
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
.
.
.
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000001000
SigCgt: 0000000000016003
CapInh: 0000000000000000
CapPrm: 00000000ffffeff
CapEff: 00000000ffffeff
```

7. Whitelist

Das Whitelist-Verfahren

Durch das Whitelist-Verfahren wird ein vollkommener Schutz gegen Viren und Fremdprogramme/Trojanische Pferde erreicht.

Das System wird in einen definierten Zustand gebracht. Zu diesem Zeitpunkt befindet sich das System noch nicht online bzw. im Produktionszustand. Von allen ausführbaren Dateien wird eine Liste (die sogen. Whitelist) bestehend aus Programmname und Checksumme erstellt.

Die Checksumme wird durch eine stark kollisionsfreie Hashfunktion erzeugt. Zur Anwendung kommt hier der Message Digest Algorithmus (MD5) mit 128 Bit oder der Secure Hash Algorithm (SHA1) mit 160, jedoch sind beliebige Algorithmen anwendbar. Charakteristisch für diese Funktionen sind möglichst große Bitunterschiede im Hashergebnis selbst bei geringen Unterschieden in der Quelldatei. Es soll für einen Angreifer unmöglich sein, eine funktionsfähige Programmdatei zu erstellen, die ein identisches Hashergebnis wie eine bereits registrierte Programmdatei hat. Stark kollisionsfreie Hashfunktionen erfüllen diese Vorgaben durch die Verwendung geeigneter mathematischer Algorithmen. Es ist in jüngster Vergangenheit theoretisch gelungen, mögliche Kollisionen in MD5-Hashes zu berechnen. Die Verwendung von MD5 als Signatur für Programme kann jedoch als sicher eingestuft werden, da hier mehrere Faktoren zusätzlich eintreffen müssten (z.B. müsste eine Datei mit künstlich kollidiertem Hashwert eine gültige ausführbare Datei sein und gültige Maschinenbefehle enthalten)

Zur Laufzeit wird jedes Programm vor der Ausführung durch das System anhand der als Meta-Information im Dateisystem abgelegten Hashs geprüft. Fehlende Einträge oder falsche Checksummen verhindern eine Erzeugung des Prozesses. Alle Programme, die nach aktivieren des Whitelist-Schutzmechanismus über beliebige Wege auf das System gelangen, können nicht ausgeführt werden. Viren und Trojanischen Pferden ist es somit unmöglich Schaden anzurichten, selbst wenn das System sie nicht als solche erkannt hat. Nur bei erfolgreichen Vergleichen wird eine Programmdatei akzeptiert.

Das Whitelist-Verfahren folgt so dem Ansatz einer regelbasierten Sicherheitsumgebung: Alles ist grundsätzlich verboten, nur über explizit definierten Regeln sind Ausnahmen möglich.

8. Verbosity

Die Verbosity, also das Mitteilungsbedürfnis, des H S S Sicherheitsmoduls lässt sich in den Abstufungen 0 - 3 über einen **Modulparameter** einstellen.

Es folgt eine Übersicht welche Meldungen in welcher Verbosity-Stufe auftreten können und was sie bedeuten.

Systemmeldungen

- H S S : Failure unregistering with the kernel - Interner Fehler, das Modul konnte nicht entladen werden. Das System neu starten.
- H S S : Error during initialization: rc = [%d] - Interner Fehler. Bitte mit Fehlercode dem Support melden.
- H S S : Successfully initialized - Das Module wurde erfolgreich initialisiert.
- H S S : Error registering with sysfs - Das Modul konnte sich nicht im SysFS registrieren. Möglicherweise ist der Kernel nicht H S S -fähig
- H S S : Failure registering with primary security module - Es wurde bereits ein anderes Sicherheitsmodul geladen, das inkompatibel mit dem H S S ist.
- H S S : Failure registering with the kernel - H S S Modul konnte nicht geladen werden. Möglicherweise ist der Kernel nicht H S S -fähig
- H S S : Error processing the password module parameter(s): rc =[%d] - Interner Fehler. Bitte mit Fehlercode dem Support melden.
- H S S : Bad verbosity [%d] must be between 0 (quiet) and 3 (talk too much) - Der Verbosity Parameter wurde auf einen ungültigen Wert gesetzt.
- H S S : Error [%d] registering hss_seclvl subsystem - Interner Fehler. Bitte mit Fehlercode dem Support melden.
- (received [%d] bytes; expected [%d] for the hexadecimal representation of the SHA1 hash of the password)
- (invalid hardware token)
- (error hashing password: rc = [%d])
- (failed to load transform for SHA1) - Es wurde kein SHA1 Support im Kernel gefunden
- (plaintext password too large (%d characters). largest possible is %lu bytes) - Es wurde versucht ein zu langes Passwort einzugeben.



- (exec of %s denied: md5 is %s but should be %s) - Die Ausführung des angegebenen Programms wurde verweigert, weil die aktuelle Signatur nicht mit der Signatur im Sicherheitsattribut übereinstimmt.
- (denied to mount to an immutable mountpoint) - Es wurde abgelehnt ein Dateisystem einzuhängen (mount), da der Einhängepunkt als IMMUTABLE markiert ist.
- (denied to umount IMMUTABLE mountpoint) - Es wurde abgelehnt ein Dateisystem abzuhängen (umount), da der Einhängepunkt als IMMUTABLE markiert ist.
- (denied to modify SUID or SGID bit) - Es wurde abgelehnt bei einer Datei das SUID bzw. SGID Flag zu setzen.
- denied writing to mounted block device) – RAW-I/O auf ein eingehängtes Dateisystem wurde verweigert.
- (denied writing to block device) - RAW-I/O auf ein Dateisystem wurde verweigert.
- (denied to decrement time) - Es wurde abgelehnt die Systemzeit zurückzusetzen.
- (denied to perform module operation) - Es wurde abgelehnt, Kernel Module zu laden oder zu entladen.
- (denied to modify IMMUTABLE/APPEND flag) - Es wurde abgelehnt, IMMUTABLE oder APPEND Datei Flags zu manipulieren.
- (denied to ptrace init process) - Ptrace des init Prozesses wurde abgelehnt.
- (%s: security.cap: %x) - Es wurde zu einer ausführbaren Datei ein CAP Sicherheitsattribut gefunden und hier angezeigt.
- (exec of %s denied: malformed or no md5 found) - Die Ausführung eines Programms wurde verhindert, weil keine gültige Signatur im Sicherheitsattribut gefunden wurde.
- (exec of %s denied: operation not supported by filesystem) - Die Ausführung eines Programms wurde verhindert, weil das Dateisystem auf dem sich das Programm befindet, keine Sicherheitsattribute unterstützt und daher keine gültige Signatur vorhanden ist. Das ist z.B. bei FAT16/32 oder ISO9660 Dateisystemen der Fall.
- (denied to mount) - Einhängen eines Dateisystems wurde abgelehnt.
- (denied to unmount) - Abhängen (umount) eines Dateisystems wurde abgelehnt.
- (denied to perform raw socket operation) – Netzwerk RAW-I/O wurde abgelehnt.
- (denied to reboot) - Reboot des System wurde abgelehnt, benutzen Sie **halt**.
- (denied DAC override) - Die Administratorfähigkeit, beliebige Dateien zu manipulieren, wurden abgelehnt.
- (denied DAC read/search override) - Die Administratorfähigkeit, beliebige Dateien zu manipulieren, wurden abgelehnt.
- (denied to perform network administrative task) - Verändern der Netzwerkeinstellungen wurde abgelehnt.
- (denied to perform raw I/O) - RAW-I/O wurde abgelehnt.
- (exec %s is not immutable, e_uid = %d, e_gid = %d) - Warnung, dass eine ausführbare Datei kein IMMUTABLE Flag hat.
- (denied capability [%d] for %s [%d] cap: %d) - Warnung, dass eine bestimmte Fähigkeit einem Prozess verweigert wurde.