Überblick des Projekts

Angesichts der realen Bedrohung durch Maleware (Viren, Trojaner, Rootkits), der zu erwartenden Bedrohung der Privatsphäre in Verbindung mit einer Aushöhlung von Grundrechten durch die Regierung und der zunehmenden Mobilität, sowohl der eingesetzten Hardware als auch der Benutzer, müssen an die Absicherung gegen unbefugten und ungewollten Zugriff auf die persönlichen Daten neue Anforderungen gestellt werden. Um diesen vielfältigen Bedrohungen zu begegnen, ist die Idee entstanden, zwei vorhandene und bekannte Techniken zu kombinieren. Die Kombination soll den ungewollten Zugriff unterbinden oder zumindest aber deutlich zu erschweren, ohne das der Benutzer das System jeweils neu konfigurieren muß. Denn, das größte Sicherheitsrisiko ist der Benutzer, bzw. dessen Bequemlichkeit.

Die hier vorgestellte Idee ist eine, von einem Wechselmedium bootbare, Firewall, die das lokal installierte OS in eine Virtualsierungsumgebung lädt.

Zuerst ein paar Grundlagen.

TCP/IP v4

1969:
Die aktuellen Sicherheitsprobleme sind historischer Ursprungs und haben Ihre Ursache im Jahre 1969. Im Jahre 1969 stieß das amerikanische Verteidigungsministerium die Vernetzung, der über die gesamte USA verteilten, Computersysteme an. Wichtig war dabei nicht nur die Kommunikation der Rechner-Systeme untereinander, sondern auch die Kommunikationsfähigkeit bei Teilausfällen des Netzwerks.

Diese war die Geburtsstunde des ARPANET und des TCP/IP-Vorläufers NCP. Das Network-Control-Protocol (NCP) wurde als Host-to-Host-Protokoll entwickelt. Zeitgleich wurden die Software-Komponenten für das Netzwerk von Bell Labs und AT&T entwickelt und bildeten die Grundlage der heutigen Unixe, inkl. Linux.

1976:
1976 fing man an auf Grundlage der Erfahrungen mit dem ARPANET und dem NCP ein neues Protokoll zu entwickeln, das Transmission Control Protocol / Internet Protocol. Hauptaugenmerk lag, ob der damaligen Leitungsgeschwindigkeiten und -qualitäten, auf der Sicherstellung das Daten sicher übertragen wurden, sprich nicht verloren gingen. Da alle Rechner einem Verbund angehörten und gegenseitig als trusted galten, waren die Mechanismen zur Absicherung der Verbindung und zur Authentifizierung der Benutzer nur schwach ausgeprägt.

1983:
Im Jahre 1983 bestand das ARPANET aus mehreren Teilnetzen, ausschließlich wissenschaftlicher und/oder militärischer Natur, hatte aber an Umfang deutlich zugenommen und wurde daher auf das neue Protokoll TCP/IP umgestellt.

Ab 1983 wird verstärkt vom Internet geredet.

1987:
1987 unterstützt IBM als erster großer Hersteller TCP/IP, neben den eigenen Protokoll-Implementierungen. Es folgen weitere Hersteller.

1991:
Der damals marktbeherrschende Hersteller von PC-Netzwerk-Betriebsystemen, Novell, unterstützt in Netware ab 1991 TCP/IP, obwohl er mit IPX/SPX ein ähnlich funktionierendes eigenes Protokoll hat.

1992:
Mehr als 10.000 Firmen nutzen das Internet.

Als vor 31 Jahren TCP/IP aus der Taufe gehoben wurde, hat niemand die Entwicklung vorhergesehen, das Rechnersysteme sich der Art verbreiten und in nahezu jeden Bereich des Lebens vordringen würden. Es gab damals weder PCs (der erste PC war der Apple II von 1977, 1981 folgte IBM PC), noch GUIs (1984 die erste GUI der Firma Apple), Festplatten gab es zu dem Zeitpunkt in der Größe von 5 MB im handlichen 8 Zoll-Format. Oder anders ausgedrückt, für die Leistungsfähigkeit eines heutigen Subnotebooks hätte man vor 31 Jahren ganze Etagen mit Elektronik vollstopfen müssen.

Zeitlich betrachtet stammt TCP/IP aus der Jungsteinzeit der Informationsverarbeitung, und man könnte als Analogie ausführen, das die TCP/IP-Kommunikation in etwa dem entspricht, als würden wir heute noch mit Trommeln, Rauchzeichen und Boten Nachrichten übermitteln. Wie man später sehen wird, greift die Analogie weiter als zunächst vermutet.

Der Adressraum von TCP/IP umfasst etwas über 4 Milliarden Einzeladressen, zum Zeitpunkt der Verabschiedung bedeutet das, eine TCP/IP-Adresse je Einwohner der gesamten Erde. Trotz des insgesamt weitsichtigen Designs des Protokolls, hat es ein paar Geburtsfehler, die im Endeffekt dazu führen, das heute nicht wenige Menschen in Lohn und Brot stehen.

Zunächst ist anzumerken, das TCP/IP ein Protokoll ist, das Daten in Paketen überträgt und zwar zwischen zwei Hosts oder zwischen einem Host und vielen anderen. Bei TCP/IP bezeichnet ein Host ein beliebiges Gerät, das eine TCP/IP-Adresse hat. Um große Mengen Daten zu übertragen, werden diese in viele kleine Pakete zerlegt, also fragmentiert, durchnummeriert und übertragen. Am anderen Ende der Verbindung werden die Daten anhand der Nummern sortiert und wieder zusammen gesetzt.

Diese relativ aufwendige Verarbeitung hat seine Ursache darin, das der Absender den Weg der Daten zum Empfänger in der Regel nicht kennt. Ähnlich wie bei einem Brief kennt der Absender der Daten nur die Adresse des Empfängers. Also versieht der Absender jedes seiner Datenpakete mit einer Absenderadresse, der Empfängeradresse und einer sequentiellen Nummer und schickt die Pakete auf die Reise.

Der nächste Host prüft anhand der Empfänger Adresse ob dies Paket für ihn ist, wenn nicht, leitet er es an eine ihm bekannte Adresse weiter.

Die Informationen Empfänger- und Absenderadresse und Sequenznummer (bestehend aus Identification und Offset) sind im TCP/IP-Header eines jeden Datenpakets enthalten, daneben enthält der Header zusätzliche Informationen, zum Beispiel die maximal Größe von Datenpaketen (Packet-Length), Time to Live und Protokoll.

Für ein abgeschottetes Netzwerk, selbst von Millionen Hosts, ein durchaus sinnvolles Protokoll, wobei es sich bei TCP/IP nicht um EIN Protokoll handelt, sondern um eine Sammlung von Protokollen für ganz unterschiedliche Zwecke. TCP/IP deckt die Layer 3-7 des ISO/OSI-Schichtenmodells ab, sprich alle Bereiche oberhalb der Schichten Physikalisch (Physical) und Datensicherung (Link-Control).

Trotz aller Beschränkungen und Geburtsfehler ist TCP/IP v4 immer noch das beherrschende Protokoll, obwohl der Nachfolger (IP v6) bereits verabschiedet und standardisiert ist.

TCP/IP ist historisch bedingt nicht besonders gut gegen Manipulationen gesichert, so ist es zum Beispiel kein Problem, die Absenderadresse von Datenpaketen zu verändern, jede NAT (Network-Address-Translation) tut dies.

Es gibt für NAT einen gewichtigen Grund, große Adressbereiche von TCP/IP, die im Internet nicht geroutet bzw. reserviert sind. Reserviert sind:

0.0.0.0 /8	aktuelles Netzwerk (/8 = Subnet 255.0.0.0)
10.0.0.0 /8	privates Netzwerk
14.0.0.0 /8	öffentliches Datennetzwerk
39.0.0.0 /8	Reserviert
127.0.0.0 /8	Lokales Netz
128.0.0.0 /16	Reserviert (/16 = Subnet 255.255.0.0)
169.254.0.0 / 16	Zeroconf
172.16.0.0. /12	Privates Netzwerk (/12 = Subnet 255.240.0.0)
191.255.0.0 /16	durch die IANA reserviert
192.0.0.0 /24	durch die IANA reserviert (/24 = Subnet 255.255.255.0)
192.0.2.0 /24	das Testnetzwerk
192.88.99.0 /24	IP6 über IP4-Tunnel und Anycast-Netzwerke
192.168.0.0 /16	Privates Netzwerk
198.18.0.0 /15	Benchmark (/15 = Subnet 255.254.0.0.)
223.255.255.0 /24	Reserviert
224.0.0.0 /4	Multicast (/4 = Subnet 240.0.0.0)
240.0.0.0 /4	Reserviert
255.255.255.255	Broadcast

Die NAT ersetzt die nicht route-bare Absenderadresse eines Hosts gegen eine route-bare, meist die des eigenen externen Interfaces.

Grundlagen Firewalling mit netfilter/iptables

Die NAT stellt die unterste und einfachste Sicherheitsstufe da. Motto: Wessen Adresse ich nicht kenne, den kann ich auch nicht erreichen.

Es soll gerüchteweise immer noch Leute geben, die sich ohne Router und/oder Firewall direkt mit dem Internet verbinden und sich eine Adresse zuweisen lassen, nur um sich nach einigen Minuten über Viren, Trojaner, Backdoors und ähnlichen Schmutz zu wundern, in naher Zukunft dürften sich dann auch noch ein paar der Drei-Buchstaben-Behörden auf dem Rechner wiederfinden, LKA, BKA, MAD und BND. Besonders verbreitet soll dies Verhalten bei Benutzern eines bestimmten Betriebsystems sein.

Eine NAT ersetzt also die Absender-Adresse eines privaten Netzwerks gegen eine, die im Internet geroutet wird. Die Antwort eines Internetservers erfolgt dann immer an die extern bekannte Adresse. Damit die Antwort nun den richtigen Rechner im Privaten Netz erreicht, muß die NAT natürlich wissen, welcher Rechner gemeint ist. Also führt die NAT interne Tabellen darüber, welcher Rechner des privaten Netzes, welche Verbindung unterhält, um eingehende Pakete zu ordnen zu können.

Wenn man also schon den Aufwand betreibt, ein Gerät mit zwei Netzwerkschnittstellen vorzuhalten, sowie jedes eingehende Paket mittels einer Software zu analysieren, liegt es nahe, auf diesem Gerät auch weitergehende Tests an den eingehenden Paketen durchzuführen. So wird aus einem einfachen Router, eine Firewall.

Linux ist ein, von Minix (einem Unix-Derivat) abgeleitetes, Betriebsystem und damit mehrbenutzerfähig, historisch bedingt, wie vorhin erwähnt, war / ist TCP/IP das native Netzwerkprotokoll aller Unicies. Somit ist ein Unix schon von sich heraus in der Lage eine Firewall zu bilden, mit der quelloffenen Unix-Variante von Linus Torwalds, eben Linux, gab es dann die Möglichkeit der Firewall für jedermann. Hier sei angemerkt, das Linux die erste, auch kommerziell, erfolgreiche Portierung eines Unix auf die PC-Hardware darstellt.

Linux besitzt seit der Kernel-Version 1.0 eine Software zur Filterung von Netzwerkpaketen, die ursprünglich von BSD stammt und im Laufe der Jahre weiterentwickelt wurde. Heute besteht die Filter-Software aus zwei Teilen:

netfilter und iptables,

wobei oft nur von netfilter oder iptables die Rede ist, jedoch immer beides gemeint wird. netfilter ist der Teil des Gespanns, der im Kernelspace liegt und iptables ist das Pendant im Userspace von Linux.

netfilter/iptables beherrscht praktisch alle relevanten TCP/IP-Protokolle und sichert, beginnend bei der Network-Schicht bis teilweise zur Applikationsschicht hinauf, die Kommunikation ab. Der Hauptteil der Absicherung findet jedoch auf den OSI-Ebenen Network und Transport statt. Beachtlich ist insbesondere der modulare Aufbau, sowohl der netfilter-, als auch der iptables-Komponente.

netfilter selbst verteilt sich über mehrere Module im Kernel, die sich jeweils ein und ausschalten lassen, so das bereits im Kernel eine gewisse Gewichtung vorgenommen werden kann.

Der hier weitaus interessantere Teil ist iptables. iptables kann durch Plug-Ins bis zum Contentfiltering aufgebohrt werden. Die Regeldefinition für eine iptables-basierte Firewall erfolgt meist über Bash-Scripte, innerhalb der Bash-Scripte sind eigene Definitionen möglich, um zum Beispiel die Verarbeitungsgeschwindigkeit zu erhöhen, in dem man selten benötigte und aufwendige Regeln verlagert. Es befinden sich zu dem eine Menge Front-Ends im Umlauf, die von der Shell bis zur aufwendigen GUI alles abdecken, Ipcop und Webmin sind zwei bekannte Vertreter.

Der grundsätzliche Aufbau von iptables ist wie folgt:

Drei Tabellen

nat	Network-Address-Translation-Tabelle
filter	Paketfiltering (die Default-Table)
mangle	Tabelle zur Manipulation von Datenpaketen

Fünf vordefinierten Ketten

Prerouting	Vor einer Routing-Entscheidung (nur **nat** + **mangle**)
Input	Pakete für den lokalen Prozess (nur **filter** + **mangle**)
Output	Pakete von dem lokalen Prozess
Forward	Pakete die geroutet werden (nur **filter** + **mangle**) [Besonderheit]
Postrouting	Alle Pakete (nur **nat** + **mangle**)

Vier Ziele

ACCEPT	Paket wird angenommen (Default)
DROP	Paket wird ohne Meldung verworfen
QUEUE	Paket geht an eine Anwendung
RETURN	Für Benutzerdefinierte Ketten als Rücksprung

Eine Menge erweiterter Ziele, die zwei Wichtigsten sind

LOG	Paket wird im Systemlog vermerkt und dann weiterverarbeitet
REJECT	Paket wird mit Rückmeldung verworfen (ICMP oder RST)

Weiterhin sind vordefinierte Muster vorhanden

tcp	TCP-Pakete auswerten
udp	UDP-Pakete auswerten
icmp	ICMP-Pakete auswerten
mac-source	MAC-Adressen filtern

Es gibt zusätzlich matches die über einen Schalter aufgerufen werden können und für die nat-Tabelle noch zusätzlichen Ketten, die da lauten

DNAT	Destination-NAT (für **Prerouting** + **Output**)
SNAT	Source NAT (für **Postrouting**)
MASQUERADE	Spezialform von SNAT (für **Postrouting**)
REDIRECT	Weiterleitung an einen lokalen Rechner, z.B. einen Proxy

Über DNAT, SNAT und MASQUERADE läßt sich ein connection-tracking etablieren, da sich netfilter die Verbindungen merkt.

Der größte Vorteil von iptables ist jedoch, das man eigene Ketten definieren kann und damit ein Regelwerk skalieren kann. Trotz umfangreicher Regelungen arbeiten so skalierte Firewalls sehr schnell. Einen zusätzlichen Geschwindigkeitsvorteil bietet die Forward-Kette, da zu routende Pakete nur ein Mal die Regeln durchlaufen müssen.

Die Regeln in der Kette werden immer von oben nach unten abgearbeitet und zwar so lange bis eine Regel in der Kette zutrifft. Trifft keine Regel zu, so wird die Standard-Policy der Kette angewendet. Die Policy ist durch ein einzelnes Ziel definiert.

Um es vereinfacht auszudrücken,
die Ketten (chains) bestimmen wo ein Paket geprüft wird,
die Muster (patterns) bestimmen welches Paket geprüft wird und
die Ziele (targets) bestimmen was mit dem Paket passieren soll.

Um eine absolute sichere Firewall mit iptables/netfilter zu definieren, braucht es nur 3 Regeln.

```
/usr/sbin/iptables -P INPUT DROP
/usr/sbin/iptables -P FORWARD DROP
/usr/sbin/iptables -P OUTPUT DROP
```

-P = Standardpolicy der Kette.

Diese Form der sicheren Firewall hat nur einen Nachteil, sie verhindert jeglichen Verkehr auf jedem Interface.

Ein einfaches Regelbeispiel, das allen Rechnern im LAN HTTP-Verbindung zu externen Web-Servern erlaubt:

```
$FW -A in2ext -p TCP --sport "$HIGHP" --dport 80 -j ACCEPT
```

$FW	ist eine Variable, steht für den Aufruf **/usr/sbin/iptables**
-A	definiert die Kette, die verwendet werden soll
in2ext	ist eine selbst definierte Kette, für FORWARD von intern nach extern
-p	definiert das Protokoll, auf das die Regel angewendet werden soll
--sport	ist ein Pattern für den Quellport der Verbindung
$HIGHP	ist eine Variable, die den TCP/IP-Portbereich 1024:65535 festlegt
--dport	ist ein Pattern für den Zielport der Verbindung
80	ist die Portnummer, auf der die HTTP-Server normalerweise lauschen
-j	ist das Ziel das verwendet werden soll
ACCEPT	das Ziel, das die Firewall anweist, das Paket zu akzeptieren

Ausgeschrieben würde die Regel wie folgt aussehen

```
/usr/sbin/iptables -t filter -A FORWARD -i 192.168.1.1 -o AAA.BBB.CCC.DDD // -->
-s 192.168.1.0/255.255.255.0 -p tcp --sport 1024:65535 --dport 80 -j ACCEPT
```

Oder auf Deutsch:

Nimm die Tabelle filter (-t).
Benutze die Kette Forward
Wenn das Paket auf Deinem Interface 192.168.1.1 (-i) ankommt und auf Deinem Interface AAA.BBB.CCC.DDD (-o) ausgegeben werden soll und diese Paket aus dem lokalen Netz 192.168.1.0/24 (-s) stammt und ein TCP-Paket ist, dann prüfe, ob das Paket von einem hohen Port an Port 80 gesendet wurde.
Trifft dies alles zu, akzeptiere das Paket und leite es weiter.

Stimmt nur eine der angegeben Bedingungen nicht, werden die nachfolgenden Regelanweisungen abgeprüft, trifft davon keine zu, so wird die Standard-Policiy verwendet, die eigentlich DROP lauten sollte.

Es ist eine ziemlich gute Idee, bei einer Firewall erst ein Mal alles zu verbieten, dies macht man mit den drei Statements der oben erwähnten sehr sicheren Firewall, um dann danach die einzelnen Verbindungen unter bestimmten Bedingungen zu erlauben.

Das obere Regelbeispiel zeigt, wie man sich über Bash-Scripte einen Haufen Tipp-Arbeit sparen kann, durch einfache Variablen-Deklaration. Durch die Definition der eigenen Kette in2ext kann man den Schreibaufwand weiter reduzieren und das Script als solches übersichtlicher gestalten.

Es ist ebenso möglich selbstdefinierte Ketten als Ziel zu deklarieren und so seltene und/oder aufwendige Paket-Analysen zu verzweigen. Das führt letztlich dazu, das die Regelabarbeitung im Standardfall schneller von statten geht.

Ein tiefergehender Ausblick in die Programmierung von iptables würde den Rahmen des Vortrages bei Weitem sprengen, so das die Möglichkeiten wie statefull-inspection, matching, loadbalancing und port-forwarding heute außen vor bleiben müssen. Aus Gründen der Übersicht und des Verständnisse sei der Weg eines Paketes durch eine netfilter / iptables Firewall anhand der obigen Regel schematisch aufgezeigt.

Wer mehr über dieses Thema wissen möchte, findet Informationen unter:

www.netfilter.org
www.iptables.org

Auch das Büchlein

„Linux iptables KURZ&GUT“ Gregor N. Purdy, O´Reilly-Verlag

hilft weiter.

Die Eine-NIC-Firewalls auf dem Heimat-Notebook arbeiten nicht anders, allerdings sind Sicherheit und Performance gegenüber der getrennt installierten Firewall beeinträchtigt.

Grundlagen Virtualisierung

Es ist ein Allgemeinplatz, das die CPUs, selbst in Desktoprechnern, hauptsächlich Däumchen drehen, weil, zu über 90% ihrer Zeit sind sie mit nichts beschäftigt, ausser darauf zu warten, daß die Schnittstelle zwischen Schreibtischstuhl und Tastatur ihnen endlich was zu tun gibt. Das führte zu dem nahe liegenden Versuch, die vorhandene aber ungenutzte Rechenleistung anderweitig zu beschäftigen.

SETI ist so ein Versuch.

Virtualisierung ist ein weiterer Versuch, auf Großrechnern schon ein alter Hut. Im Großrechner- oder Mainframe-Bereich wird meist Partionierung auf Hardwareebene betrieben.

Auch im Bereich x86-Architektur ist Virtualisierung nicht so ganz neu, allerdings findet hier die Virtualisierung meist mittels Software statt. Bekannte Vertreter sind VMware und Virtual-PC in letzter Zeit sind noch Xen, QEMU und Virtual-Box hinzugekommen. Relativ neu ist im x86-Bereich die Virtualisiierung auf Prozessor-Ebene. Die Technik ist seit Ende 2005 (intel) bzw. Mitte 2006 (AMD) verfügbar und basiert auf Erweiterungen des Prozessor-Befehlssatzes der eine so genannte Secure-Virtual-Machine implementiert. Natürlich sind die Befehlssätze inkompatibel zueinander.

Die Virtualisierung auf Prozessorebene hat den Vorteil der höheren Geschwindigkeit.

Da es mehrere Ansätze zur Virtualisierung mittels Software gibt, sollen die Unterschiede kurz vorgestellt werden.

Eine weitverbreitete Form der Virtualisierung ist die der BSD-Jails oder des VServers. Hier wird Anwendungen eine komplette, vollständig abgeschottete Laufzeitumgebung vom Betriebsystem zur Verfügung gestellt. In dieser Laufzeitumgebung, auch als Container bezeichnet, kann die Anwendung arbeiten, als wäre sie alleine auf dem System, allerdings steht ihr auch nur eine Teilmenge des Systems zur Verfügung. Andere Betriebsysteme lassen sich nicht laden, das ursprünglich gebootete Betriebsystem ist das einzige, das läuft.

Eine andere Möglichkeit der Virtualisierung ist die Systemvirtualisierung, die es ermöglicht unterschiedliche Betriebsysteme zu laden. Hierbei wird jedem Gast-Betriebsystem praktisch ein virtueller PC zur Verfügung gestellt. Um dies zu ermöglichen wird zwischen Hardware und Betriebsystemkern eine Zwischenschicht eingezogen, ersatzweise auch zwischen einem Host-OS und einem Gast-OS.

Bei der Systemvirtualisierung gibt es mehrere Lösungen, die sich im Grunde nur durch den Grad der Hardware-Emulation unterscheiden.

Zu erst wäre da die vollständige Hardwarevirtualisierung, daß heißt jedem geladenen OS wird ein komplettes Hardware-System emuliert. Daher auch

Hardware-Emulation.

Die Hardware-Emulation ermöglicht es, nicht nur beliebige Betriebsysteme für die reale Hardware des Host-Systems zu laden, sondern auch Betriebsysteme für nahezu beliebige andere Hardware, sofern die Emulation die Hardware-Plattform unterstützt.

Die hohe Flexibilität bei der Virtualisierung wird mit Geschwindigkeitseinbußen und einem hohen Hardware-Einsatz auf dem Host-System erkauft, sowie einem erheblichen Verwaltungsaufwand beim Hypervisior (der Zwischenschicht). Vorteil ist, das alle Betriebsysteme und Anwendngen für die emuliert Hardware auf der emulierten Hardware ohne jede Änderung benutzt werden können.

Die nächste Stufe der Systemvirtualisierung heißt dann wirklich

Hardware-Virtualisierung

Hierbei wird dem Gast-Betriebsystem ein Teilbereich der ursprünglichen Hardware emuliert und es läuft in einer abgeschotteten Umgebung. Mit dem niedrigeren Emulationsaufwand steigt natürlich die Geschwindigkeit und es sinken die Hardware-Anforderungen für den Host-Rechner.

Es können nur Betriebsysteme virtuell zur Verfügung gestellt werden, die nativ für die Hardware programmiert wurden. Auch für diesen Bereich müssen keine Anpassung des virtualisierten Betriebsystems vorgenommen werden. Dies ist auch der Bereich in dem die Virtualisierungsbefehlssätze der o.a. x86-Prozessoren ihre Vorteile ausspielen.

Die dritte Form der Systemvirtualisierung wird als

Paravirtualisierung

bezeichnet. Bei der Paravirtualisierung wird keinerlei Hardware emuliert, sondern die vorhandene Hardware wird abstrahiert und die Gast-Betriebsysteme greifen auf diese Abstraktionsschicht zu. Dies bedingt eine sehr schlanke VM und es gibt kaum merkliche Geschwindigkeitseinbußen. Allerdings müssen die Betriebsysteme auf die VM angepasst werden.

Meine Wahl ist auf Xen gefallen, da es für das Projekt einige Merkmale aufweist, die vorteilhaft erscheinen.

1. Xen kommt aus dem universitären Umfeld (Cambridge)
2. Xen steht unter der GPL
3. Xen ist für die wichtigsten Distributionen (SuSE, Fedora, Debian) verfügbar, bzw. integriert
4. Xen ist ein ausgereiftes Produkt
5. Xen ist servertauglich (SELS, RHEL, Netware)
6. Xen unterstützt die Virtualsierungstechniken von intel + AMD (darauf wird hier jedoch nicht näher eingegangen)
7. Xen wird von Unternehmen wie SUN, IBM, Novell, Redhat, Microsoft, IBM, AMD und intel unterstützt
8. Es gibt Bemühungen Xen als Industriestandard zu etablieren

Xen unterscheidet die Betriebsysteme als Domainen, dabei fällt der Domain 0 (genannt dom0) eine besondere Rolle zu, sie kontrolliert die anderen Domainen, (genannt domU), bzw. die Virtualisierungsschicht, den Hypervisor.

Bei Xen wird zwischen die Hardware und alle Betriebsysteme eine Zwischenschicht eingezogen, der Hypervisor. Bei den Betriebsystemen wird zwischen Privilegierten (dom0) und Unprivilegierten unterschieden.

dom0	Priviligiert	Wirt
domU	Unpriviligiert	Gast

Es gibt nur eine dom0 aber beliebig viele, nur durch die Resourcen begrenzte, domU, diese Eigenschaft macht Xen besonders interessant für das Projekt.

Mehr zum Thema:
cl.cam.ac.uk/research/srg/netos/xen/
„Xen – Virtualisierung unter Linux“, Opensourcepress ISBN 978-3-937514-29-1

Das Projekt

Die alltäglichen Gefahren, wie Software-Lücken, Viren, Würmer, Trojaner, backdoors und rootkits, lasse ich außen vor, sie haben bei meinen Überlegungen und der eigentlichen Idee keine Rolle gespielt, obwohl sie natürlich auch eine potentielle Bedrohung darstellen.

Alltags-Gefahren

Zu meinem Alltag gehörte und gehört folgende Situation:

Ich sitze mit meinem Notebook in meinem Heimat-Büro, abgeschirmt gegen das Internet mit einer extra Firewall und einem Router, feste IP-Adressen im LAN, der Serveradmin bin ich, die User in meinem Netzwerk kenne ich persönlich. Eine Verbindung die ich nicht zulasse, geht nicht ins Internet oder auf einen anderen Rechner.

Es kommt der Anruf eines Kunden, ich packe mein Notebook ein, fahre zum Kunden und stöpsel mich dort ans Netz. Meist muß ich erst mal auf DHCP umschalten, weil selbst 3-Mann-Firmen inzwischen DHCP-Netzwerke betreiben. Gegen das Internet bin ich abgeschirmt. Aber gegen den Rest?

Wer sagt, das die bösen Buben nur im Internet lauern?

Hier im Raum ist genügend Wissen vorhanden, um sich auf jeden ungeschützten Rechner im lokalen Netz einzuhacken. Nicht, das dies einem der Anwesenden unterstellt werden soll. Warum soll dieses Wissen nicht auch bei meinem Kunden vorhanden sein?

Nun werden sicher einige einwenden, sie hätten eine lokale Firewall aktiviert. Aber Port 3333 oder 3306 (MySQL) deaktiviert, Port 3000 (HBCI) nicht vergessen, auch 23 (telnet) abgeschaltet und 123 (ntp) nicht übersehen?

Nun werden einige einwenden, sie hätten ihr Home zusätzlich noch verschlüsselt. Das schütz aber weder gegen Keylogger noch gegen eine Löschung.

Wieder andere werden sagen, ich habe nichts Interessantes auf meiner Festplatte. Was wohl der größte Irrtum aller Zeiten ist. 1983 befand der 1. Senat des BVG, angesichts einer relativ unbedenklichen Volkszählung, das es

KEINE

unverfänglichen Daten gibt, und kreierte das Grundrecht der informationellen Selbstbestimmung.

Ein drastisches Beispiel aus der Geschichte zu unverfänglichen Daten, ist das, der Religionszugehörigkeit, bis 1933 war das eine völlig unverfängliche Information. Nach einer demokratischen Wahl wurde aus der Information eine tödliche Bedrohung.

Diese Nebensächlichkeiten sind schon Bedrohung genug und führten bei mir bereits vor einigen Jahren zu Überlegungen in Richtung dieses Projekts.

Politische Lage

Spannend wird die potentielle Bedrohung, seit dieser berufsparanoide Schwabe mit der Rückendeckung der ehemaligen **FDJ-Sekretärin für Agitation und Propaganda** laut darüber nachdenken darf, das ein Verdacht ausreichen könnte, um bei jeden beliebigen Bürger beliebig weit und unkontrolliert in die Intimsphäre einzudringen. Sprich, setzt Schäuble sich durch, sind die ersten 20 Artikel des Grundgesetzes das Papier nicht wert, auf dem sie gedruckt werden.

Da nun auf einen groben Klotz ein grober Keil gehört, könnte man über Scherze nachdenken, wie eine, an einen Bewegungsmelder gekoppelte WebCam, die immer dann scharf geschaltet wird, wenn man sein Haus verlässt und die die Bilder potentieller Eindringlinge direkt auf einen Server irgendwo in Deutschland überträgt. Wäre natürlich nett, wenn man Bilder der Personen veröffentlicht, die einem den Bundestrojaner installieren oder das System ausforschen.

Aber was ist, wenn die nie kommen. Ein paar hundert Euro in den Sand gesetzt. Vielleicht finden die ja auch einen anderen Weg, z.B. über die MAC, um per Internet ans System zu kommen?

Vor allem, was bei der ganzen Diskussion nicht bedacht wird, nach der neusten Änderung des Strafgesetzbuches, darf man nicht ein Mal mehr die Programme besitzen, die es ermöglichen, eine entsprechende Manipulation durch die Behörden aufzudecken und rückgängig zu machen.

Auch wird gerne unberücksichtig gelassen, das es ein relativ einfacher Weg ist, Hersteller von Software, die nicht der GPL unterliegt, und dazu gehört auch das BIOS, zu einer Zusammenarbeit zu überreden und dort bereits backdoors einzubauen.

Damit hat meine Idee urplötzlich tagesaktuelle Brisanz erhalten, so das ich nun gerne dies auf eine breitere Basis stellen würde um daraus ein Projekt zu machen, so lange noch die Möglichkeit dazu besteht.

Projekt-Idee

Das größte Problem bei der Absicherung von Systemen ist die menschliche Bequemlichkeit, die selbst einfachste Sicherheits-Maßnahmen wie sichere Passwörter regelmäßig ins Leere laufen läßt. Da die Bequemlichkeit auch eine meiner herausstechenden Eigenschaften ist, war und ist es mir zu viel Arbeit, meinen Laptop jedes Mal mit einer lokalen Firewall zu versehen, wenn ich mich in fremde Netze einhänge.

Andererseits hat ausser mir niemand was auf meinem System zu suchen, meine Freundin nicht, meine Kunden nicht und irgendwelche Behörden schon zwei Mal nicht.

Vor 3 oder 4 Jahren hatte ich dann die Idee, das OS samt einer Firewall auf einen Wechseldatenträger zu installieren, was das eigentliche Problem nur verlagerte, da ich immer noch mindestens 2 Boot-Medien brauchte. Eins ohne Firewall für mein Heimatbüro und eins mit Firewall für unterwegs.

Also wurde das Problem vertagt.

Mit der Integration von Xen in SuSE eröffnete sich ein neuer Lösungsansatz, den ich allerdings nicht weiter verfolgte. Mit den neuseten, nur als freiheits- und demokratiefeindlich zu bezeichnenden, Diskussion wurde das Thema hoch interessant.

Angesichts der realen Bedrohung der Grundrechte kann ich mir, denke ich, eine weitergehende Begründung für das Projekt sparen.

Lösungsansätze:

In dem Projekt geht es darum, den unkontrollierten Zugriff auf den persönlichen Computer zu unterbinden, sei es in fremden Umgebungen, sei es wenn der Computer unbeaufsichtigt ist. Sprich darum, das nicht ein Mal aus Versehen jemand Zugriff auf das System erhält, gleichzeitig soll die Lösung aber die Bequemlichkeit des Benutzers nicht vollständig aushebeln oder einschränken.

Es gibt diverse Lösungsansätze, die aber jeweils nur einen Teilbereich der Problematik abdecken und zudem nicht unbedingt vertrauenswürdig sind.

Zu den Zugangsschutzmöglichkeiten gehören Fingerabdruck-Scanner, Smartcards und BIOS-Passwörter. Diese Techniken sollen den unbefugten Start des Systems verhindern sind aber proprietäre Lösungen, so das nicht nachvollziehbar ist, welche Hintertüren ggf. vorhanden sind.

Den Schutz vor Zugriff auf persönliche Daten kann man mittels Verschlüsselung des Home-Verzeichnises relativ einfach realisieren, sollte dabei aber nur offen Algorithmen verwenden.

Der Schutz vor ungewolltem Zugriff während man aktiv arbeitet, erfordert aber deutlichen Mehraufwand und auch deutlich mehr Kenntnisse.

Die Überlegung ist nun:

Das OS samt Firewall auf einen entfernbaren Datenträger zu verlegen und auf diesem auch eventuell notwendige Schlüssel zu hinterlegen. Als Datenträger kämen, aufgrund der Datenmenge, nur CD / DVD / USB-Stick oder Speicherkarten in Frage.

Auf diesem Datenträger wird ein OS mit Firewall installiert und als Wirt (dom0) für Xen konfiguriert. Das auf der lokalen Platte vorhandene OS soll (sofern es über einen Xen-tauglichen Kernel verfügt) als Gast (domU) nachgestartet werden. Das Gast-OS stellt dann den eigentlichen Arbeitsplatz zur Verfügung.

Diese Konfiguration führt dazu, das ungewollte Manipulation an dem eigentlichen Boot-OS kaum möglich sind, gleichzeitig übernimmt das Boot-OS die alleinige Kontrolle über die Hardware und teilt dem Gast-OS die verfügbaren Resourcen zu. Geht man davon aus, das es möglich ist, das Wirt-OS als monolithischen Kernel zu realisierne, das Wirt-OS von einem nicht beschreibbaren Datenträger zu booten und diesem die alleinige Kontrolle über den Hypervisor, sprich, die Hardware, zu übertragen, so führt dies dazu, daß das System praktisch manipulationssicher ist.

Xen stellt ab der Version 3.0 drei Wege zur Verfügung, den Gast-OSen einen Zugang zur NIC zu ermöglichen.

Virtuelles Bridging

Virtuelles Routing

Virtuelle NAT

Virtuelles Bridging ist dabei die Standardlösung, würde aber voraussetzen, das die Firewall auf Layer 2 etabliert werden müßte, sprich mit ebtables.

Bleiben Routing und NAT. Beide virtuellen Lösungen funktionieren wie von der physikalischen Gegebenheiten her bekannt, mit einem großen Unterschied:

Die Gast-OS (domU) haben keine MAC und sind daher von ausserhalb des Rechners nicht direkt erreichbar!

Genau das soll auch erreicht werden. Das Gast-OS ist von ausserhalb der lokalen Maschine praktisch kaum angreifbar und zusätzlich sorgt das Wirt-OS (dom0) dafür, daß das Gast-OS nicht unkontrolliert Verbindungen wohin auch immer aufbauen kann.