Universität Bern, Schweiz
Institut für Informatik und angewandte Mathematik
Vorlesung „Betriebssysteme und Verteilte Systeme“

Klassische Sicherheitslücken

David Jörg

Inhaltsverzeichnis

Abbildungsverzeichnis

Klassische Sicherheitslücken

1 Einführung

Fast täglich kann man in den einschlägigen Mailinglisten und Security-Online-Archiven Nachrichten von neuen Einbruchsmöglichkeiten und Verwundbarkeiten in Applikationen oder Betriebssystemen lesen. Wer sein System nicht auf dem aktuellsten Stand hält und über die Verwundbarkeiten informiert ist, steht bald einmal vor einem eingebrochenen System. In vielen Fällen ist die Ursache ein Pufferüberlauf, englisch Buffer Overflow.

Dieses Dokument geht einerseits kurz darauf ein, wie ein Angreifer Verwundbarkeiten und Sicherheitslücken auf Systemen sucht. Ferner wird dann auf einige wenige bekannte Sicherheitslücken bei Betriebssystemen und Netzwerken eingegangen, insbesondere Buffer Overflow und Denial of Service Attacken.

2 Finden von Sicherheitslücken

2.1 "Footprinting"

Footprinting dient dazu, ein möglichst komplettes Profil des Sicherheitskonzepts eines Unternehmens zu erhalten. Durch gezieltes Anwenden verschiedener Tools (z.B. whois, nslookup, traceroute) kann ein Angreifer den Internet-Zugang eines ganzen Unternehmens auf spezifische Bereiche von Domains, Netzwerkblöcke, einzelne IP-Adressen oder Systeme reduzieren. Im Internet gehört z.B. dazu das Herausfinden von Domain-Namen, Netzwerkblöcken, IP-Adressen von Systemen, DNS- und Mail-Server, System-Architektur (z.B. SPARC), Intrusion Detection Systemen, Zugriffsmechanismen etc. In Intranets interessieren zum Beipiel zusätzlich die verwendeten Protokolle (IP, IPX, DecNET etc.).

2.2 Scanning

In einer zweiten Phase geht es darum herauszufinden, welche Systeme laufen und erreichbar sind, zudem sucht man auf den erreichbaren Systemen auch nach offenen Ports. Anhand der Portnummern lässt sich dann auf die verwendeten und verfügbaren Dienste schliessen. Diese Phase wird als *Scanning* bezeichnet.

Die erreichbaren Systeme werden mit Hilfe eines *Ping Sweeps* auf einen IP-Adressblock gefunden. Ping sendet ein ICMP ECHO Datenpaket und wartet auf ein ICMP ECHO_REPLY Paket von der Zielmaschine. Wird ein solches Paket empfangen, zeigt dies, dass der entsprechende Host erreichbar ist. Ein bekanntes dazu verwendetes Tool ist der Network Mapper (nmap):

```
% nmap –sP 192.168.1.0/24

Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )

Host  (192.168.1.0) seems to be a subnet broadcast
address (returned 3 extra pings).
Host  (192.168.1.10) appears to be up.
Host  (192.168.1.11) appears to be up.
Host  (192.168.1.15) appears to be up.
Host  (192.168.1.50) appears to be up.
Host  (192.168.1.101) appears to be up.
Host  (192.168.1.255) seems to be a subnet broadcast
address (returned 3 extra pings).
Nmap run completed -- 256 IP addresses (7 hosts up) scanned in 21 seconds
```

Abbildung 1: Ping Sweep (nmap)

Obwohl ein Ping Sweep an sich ungefährlich ist, ist es wichtig, die Aktivität festzustellen, da sie auf einen bevorstehenden Angriff hindeuten kann. Um Ping Sweeps vorzubeugen gibt es unterschiedliche Massnahmen, die mit Hilfe einer Firewall durchgesetzt werden können:

- Blockieren des gesamten ICMP-Verkehrs. Dies hat den Nachteil, dass auch erlaubte ICMP-Pakete, die zu Netzwerkdiagnosezwecken versendet werden, weggefiltert werden (z.B. ping, traceroute).
- Festlegen von Access Control Lists bestimmter bekannter IP-Adressen, die ICMP Pakete durch die Firewall senden dürfen.
- Durchlassen nur bestimmter ICMP Pakete: Für eine DMZ (demilitarisierte Zone) wäre eine mögliche Konfiguration z.B. ICMP ECHO_REPLY, HOST UNREACHABLE und TIME EXCEEDED durchzulassen und alle anderen Pakete zu blockieren.

Sobald die verfügbaren Maschinen bekannt sind, geht es noch darum festzustellen, welche möglicherweise verwundbaren Netzwerkdienste darauf laufen. Dies geschieht mittels eines *Portscans*. Es gibt viele verschiedene Arten von Scans:

- **TCP connect scan:** Dieser Scantyp führt einen vollständigen Dreiwege-Handshake aus (SYN, SYN/ACK, ACK), wie er beim Aufbau einer TCP-Verbindung durchgeführt wird. Dieser Scantyp ist vom Zielsystem jedoch einfach zu erkennen.
- **TCP SYN scan:** Der TCP SYN scan ist ein halboffener Scanvorgang, da keine volle TCP-Verbindung hergestellt wird. Es wird ein SYN-Paket an den Zielport gesendet. Wird von der Zielmaschine ein SYN/ACK-Paket empfangen, befindet sich die Maschine im LISTENING Zustand und der Port ist offen. Die Portscan durchführende Maschine sendet nun ein RST/ACK-Paket, damit die TCP-Verbindung nie hergestellt wird. Diese Technik ist verdeckter als das Herstellen einer TCP-Verbindung und wird von der Zielmaschine möglicherweise nicht aufgezeichnet.
- **TCP FIN scan:** Hier wird ein FIN-Paket zum Zielport gesendet. Gemäss RFC 793 sollte das System ein RST für alle geschlossenen Ports zurücksenden.
- **Weitere Scanmethoden:** Es existieren viele weitere Methoden von möglichen Scan-Typen, auf die hier nicht weiter eingegangen wird: TCP Xmas Tree scan, TCP Null scan, TCP ACK scan, TCP Windows scan, TCP RPC scan, UDP scan.

Auch um Portscans durchzuführen kann beispielsweise der Network Mapper eingesetzt werden. Jedoch existieren auch eine Menge anderer Portscan-Utilities.

```
% nmap –I 192.168.1.1
Starting nmap V. 2.53 by fyodor@insecure.org
Port    State       Protocol  Service         Owner
22      open        tcp        ssh             root
25      open        tcp        smtp            root
80      open        tcp        http            root
110     open        tcp        pop-3           root
113     open        tcp        auth            root
6000    open        tcp        X11             root
```

Abbildung 2: Portscan (nmap)

Wie Ping Sweeps sind auch Portscans mögliche Vorboten bevorstehender Angriffe, deshalb ist es ebenfalls hier wichtig Gegenmassnahmen zu ergreifen:

- Als primäre Methode werden netzwerkbasierte Intrusion Detection Systeme (IDS) eingesetzt, welche Postscans erkennen und aufzeichnen. Heutige Firewalls bieten auch die Möglichkeit beim Erkennen eines Portscans weitere Pakete des Absenders während einer gewissen Zeit zu blockieren.
- Weiter muss eine Firewall so konfiguriert werden, dass nur Zugriff auf umbedingt notwendige Dienste gewährt wird. Zugriffe auf andere Ports werden blockiert. Für den Zugriff auf einen Webserver ist z.B. ausschliesslich Port 80 (http) und eventuell noch Port 443 (https) für verschlüsselte Verbindungen erforderlich.
- Auf den einzelnen Maschinen kann das Risiko minimiert werden, indem nicht verwendete Dienste ausgeschaltet werden. Besonders gefährlich sind Remote Dienste (z.B. rlogin, rsh, rwho, ruptime…).

Neben Portscanning existiert auch die Möglichkeit mittels *Stack fingerprinting* recht zuverlässig auf das verwendete Betriebssystem und dessen Version zu schliessen. Jeder Hersteller implementiert den TCP/IP-Netzwerkstack etwas unterschiedlich. Auch wenn die Spezifikationen in einem RFC-Dokument vorgegeben sind, gibt es noch Freiräume bei der Implementation. Anhand dieser Nuancen ist es möglich, auf das verwendete Betriebssystem zu schliessen. Dies kann auch passiv durch Mitschneiden von Datenpaketen geschehen.

3 Systeme

Bei Sicherheitslücken bei Systemen kann generell unterschieden werden zwischen lokalem und Remotezugriff. Remotezugriff beinhaltet den Zugriff via Netzwerk oder einem anderen Kommunikationsmediums. Lokaler Zugriff beinhaltet physischen Zugriff auf eine Maschine, z.B. eine Shell oder ein Login.

3.1 Local Access

3.1.1 Passwörter

Ein grosses Problem bei der Sicherheit von Systemen liegt in der schlechten Wahl von Passwörtern der Benutzer oder deren nicht Geheimhaltung. Die grundlegenste Attacke ist das simple “Erraten” von schlechten Passwörtern durch eine Brute Force Attacke. Brute Force ist eine der effektivsten Wege für Angreifer, in ein System einzubrechen.

Das Cracken von Passwörtern kann einerseits als aktiver Remoteangriff erfolgen, durch ausnutzen von Diensten, die den Benutzer authentifizieren: telnet, ftp, rlogin, rsh, ssh, SNMP community names, POP (Post Office Protocol), HTTP, HTTPS sind einige Beispiele. Ein lokaler Angriff erfolgt meist offline und passiv. Der Angreifer muss sich ausschliesslich Lesezugriff zur `/etc/passwd` oder `shadow` Datei verschaffen.

Passwort Cracking geschieht meistens als voll automatisierte Wörterbuch-Attacke. Der Angreifer verschlüsselt ein Wort oder zufälligen Text mittels dem bekannten Passwort-Hash-Algorithmus (meistens DES) und vergleicht das verschlüsselte Passwort mit den Einträgen in der Passwort-Datei. Stimmt die Ausgabe des Hash-Algorithmus mit dem verschlüsselten Passworteintrag eines Benutzers überein, wissen wir was das ursprüngliche Passwort war. Passwort Crack Programme sind hoch optimiert, kennen verschiedene Hash-Algorithmen und eine Menge von Regeln, um Permutationen von Wörtern zu erzeugen.

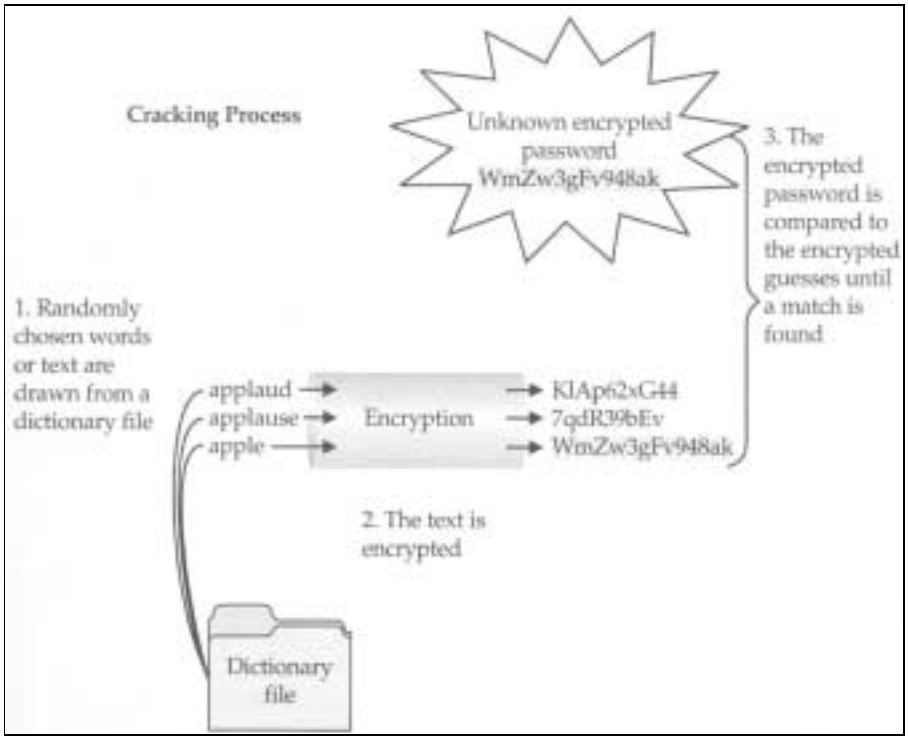


Abbildung 3: Ablauf einer Brute Force Attacke auf `/etc/passwd`

Die einfachste Gegenmassnahme gegen Brute Force Attacken ist die Wahl von sicheren, längeren Passwörtern (mind. 6 Zeichen), die nicht einfach erraten werden

können. Standardmässige Passwörter für “setup” oder “admin” Accounts sollten zwingenderweise geändert werden.

3.1.2 Symlink Attacken

Auf UNIX-Systemen werden viele temporäre Dateien in `/tmp` oder in anderen Verzeichnissen abgelegt. Auch viele SUID[1] root Programme legen temporäre Dateien an, einige führen aber nicht die kleinste Menge von Sicherheits-überprüfungen durch. Gefährlich sind vorallem symbolische Links[2] zu anderen Dateien, wie folgend an einem Beispiel beschrieben wird:
Bekannt ist z.B. die `dtappgather` Sicherheitslücke für Solaris. `Dtappgather` ist ein Utility, das standardmässig mit dem Common Desktop Environment ausgeliefert wird. Wird `dtappgather` ausgeführt, legt es eine temporäre Datei `/var/dt/appconfig/appmanager/generic-display-0` an und setzt die Berechtigungen auf 0666 (schreiben und lesen für alle). Zudem wird der Besitzer der Datei auf denjenigen Benutzer gesetzt, der das Programm ausführt. Leider überprüft `dtappgather` nicht, ob die Datei schon existiert und ob es sich um einen symbolischen Link handelt. Kreiert nun ein Angreifer einen symbolischen Link von `generic-display-0` auf die `/etc/passwd` Datei, wird beim nächsten Ausführen von `dtappgather` die Passwort-Datei auf 0666 gesetzt und der Angreifer wird Besitzer der Datei. Dasselbe ist mit `/etc/shadow` notwendig. Das Einrichten eines 0 UID (root) Accounts ist nun die Sache von weniger als einer Minute.

3.1.3 File Descriptor Attacken

Dateideskriptoren sind nichtnegative ganze Zahlen, die das System verwendet, um offene Dateien zu verwalten. Diese werden anstelle der spezifischen Dateinamen verwendet. Öffnet der Kernel eine bestehende Datei oder wird ein neues File kreiert, gibt er dem Programm einen spezifischen Dateideskriptor zurück, mit dem von der Datei gelesen oder in die Datei geschrieben werden kann. Ist ein File Descriptor für einen privilegierten Prozess offen im Lese/Schreib-Modus und allokiert dieser die File Deskriptoren nicht korrekt, ist es für einen Angreifer eventuell möglich, in die Datei hineinzuschreiben, während sie editiert wird. So kann ein Angreifer kritische Systemdateien (z.B. `/etc/passwd`) modifizieren und Administratorrechte erlangen. Die Verantwortung für File Descriptor Attacken liegt ganz klar beim Programmierer von SUID Programmen – sorgfältiges Programmieren ist unabdingbar. Zudem sollten die SUID Bits bei jedem Programm entfernt werden, wo sie nicht umbedingt erforderlich sind.

3.1.4 “Race Conditions”

Angreifer werden immer versuchen diejenige Zeit auszunutzen, während welcher ein Programm eine privilegierte Operation ausführt. Der Angriff musst so vorbereitet

[1] SUID bedeutet „Set user id root“. Dies erlaubt Programmen Funktionen auszuführen, zu denen der eigentliche Benutzer nicht berechtigt wäre. Dazu gehören low level Netzwerkbefehle, graphische Display-Funktionen, Ändern von Passwörtern usw.
[2] Ein symbolischer Link ist eine Datei, die auf eine weitere Datei zeigt. In UNIX werden symbolische Links mit dem `ln` Befehl erstellt. „`ln –s /tmp/foo /etc/passwd`“ bewirkt zum Beispiel, dass die Datei /tmp/foo auf /etc/passwd zeigt.

werden, dass er stattfindet, nachdem das Programm den privilegierten Modus betritt und bevor es seine Privilegien wieder aufgibt. Typischerweise findet dies nur während einem kurzen Zeitfenster statt. Eine Verwundbarkeit, die dieses Zeitfenster ausnutzt wird auch eine *Race condition* genannt. Es gibt viele verschiedene Typen von Race conditions, beim Signal Handling[3] können z.B. oft solche auftreten. Ein Beispiel einer Signal Handling Verwundbarkeit ist z.B. die wu-ftpd v2.4 Verwundbarkeit (http://www.cert.org/advisories/CA-1997-16.html).

3.2 Remote Access

3.2.1 Brute Force Attacken

siehe Kapitel 3.1.1 (Passwörter)

3.2.2 Buffer Overflow

Ein *Buffer Overflow* (Pufferüberlauf) geschieht dann, wenn ein Benutzer versucht, mehr Daten in in einen Puffer zu schreiben als dafür vorgesehen und allokiert ist. Dies kann häufig bei C-Programmen auftreten, die Funktionen wie `strcpy()`, `strcat()`, `sprintf()` usw. ausführen. Normalerweise führt ein Pufferüberlauf zu einem Programmabsturz (Segmentation violation). Jedoch kann dieses Verhalten ausgenutzt werden, um Zugriff zu einem Computersystem zu erlangen. Pufferüberläufe können sowohl lokal als auch remote ausgenutzt werden.

Eine detaillierte Beschreibung folgt in Kapitel 5.1 (Buffer Overflow).

3.2.3 RPC

Remote Procedure Call (RPC) erlaubt Programmen Code auf einem anderen, entfernten System auszuführen. Eine der ersten RPC Implementationen wurde von Sun Microsystems entwickelt. RPC Dienste registrieren sich beim Start beim Portmapper. Um einen RPC Dienst zu kontaktieren, muss der Client zuerst beim Portmapper anfragen, auf welchem Port sich der benötigte RPC Dienst befindet. Viele standardmässigen UNIX-Syteme starten mehrere RPC Dienste schon beim Systemstart. Zudem sind viele RPC Dienste ausserordentlich komplex und laufen mit root Privilegien. Deshalb führt ein erfolgreicher Buffer Overflow oder eine erfolgreiche Input Validation Attack sofort zu root Zugriff auf das System.

Die beste Abwehrstrategie gegen RPC Attacken ist die Abschaltung aller nicht unbedingt benötigten RPC Dienste. Ist ein RPC Dienst für ein System unabdingbar, ist es sicherer eine Zugriffskontrolle zu implementieren, die nur berechtigten Systemen Zugriff auf RPC Ports erlaubt. Das Einschalten eines nicht-ausführbaren Stacks ist bei Systemen zu empfehlen die dies erlauben – dies ist eine Masnahme

[3] Signale sind Mechanismen in UNIX, um einem Prozess mitzuteilen, dass ein bestimmter Zustand eingetreten ist und dienen auch dazu, asynchrone Ereignisse zu verarbeiten. Wenn zum Beispiel ein Benutzer ein laufendes Programm unterbrechen will, drückt er CTRL-Z. Dies sendet ein `SIGTSTP` Signal zu allen Prozessen in der Vordergrund Prozessgruppe. Signale werden also verwendet, um den Fluss eines Programms zu ändern.

gegen Buffer Overflows. Schliesslich bietet die Verwendung von Secure RPC noch die Möglichkeit, eine zusätzliche Berechtigungskontrolle einzufügen.

3.2.4 NFS

NFS (Network File System) ist ein von Sun Microsystems entwickeltes System, welches eingesetzt wird, um transparenten Zugriff auf Dateien und Verzeichnisse auf entfernten Systemen zu ermöglichen, als ob die Daten lokal gespeichert wären. Das Potential des Missbrauchs von NFS ist sehr hoch und eine der gängigen UNIX-Attacken. In den Vergangenheit wurden eine Menge Buffer Overflow Attacken auf `mountd`, den NFS-Server entdeckt. Weiter basiert NFS auf RPC Diensten und kann einfach dazu überlistet werden Angreifern das Mounten eines entfernten Dateisystems zu ermöglichen. Die grösste Teil der Sicherheit von NFS basiert auf *File Handles*. Das File Handle ist ein Token, welches jede Datei und jedes Verzeichnis auf dem entfernten System eindeutig identifiziert. Kann ein File Handle abgehört oder erraten werden, kann ein Angreifer einfach auf diese Dateien auf dem entfernten System zugreifen.

Wird NFS nicht benötigt, sollten NFS und alle dazugehörigen Dienste (z.B. `mountd`, `statd`, `lockd`) abgeschaltet werden. Client- und Benutzerzugriffskontrolle sollten implementiert werden, um nur berechtigten Benutzern Zugriff auf die benötigten Dateien zu gewähren. Es existieren Optionen, um berechtigte Maschinennamen festzulegen, Netzwerk-Gruppen, read-only Optionen und die Möglichkeit das SUID Bit zu verbieten.

3.2.5 Domain Name System Hacking

DNS ist einer der am meisten verwendeten Dienste auf dem Internet. Die Allgegenwart von DNS verleitet selbst zur Attacke. Viele Angreifer erproben routinemässig Verwundbarkeiten in der am meisten verbreiteten Implementation von DNS für UNIX, dem Berkeley Internet Name Domain (BIND) Paket. Zusätzlich ist DNS einer der wenigen Dienste, die jedes Unternehmen braucht und die vom Internet her zugänglich sein müssen. Ein Fehler in BIND führt also fast sicher zu einem kompromitierten System, meistens mit root Privilegien.

Gegenmassnahmen: BIND sollte auf jedem System, das nicht für DNS-Zwecke eingesetzt wird abgeschaltet werden. Für DNS-Server muss BIND immer auf dem neuesten Stand gehalten werden und man sollte regelmässig aktuelle Patches installieren. Weiter sollte `named` als unprivilegierter Benutzer ausgeführt werden. Zum Start sind root Privilegien erforderlich, diese können aber nach dem Binden zu Port 53 wieder abgegeben werden. Schliesslich sollte `named` mit Hilfe der `-t` Option aus einer `chrooted()` Umgebung ausgeführt werden, damit ein Angreifer selbst bei erfolgreicher Attacke nicht Zugriff auf das ganze Dateisystem erlangen kann (`named -u dns -g dns -t /home/dns`).

4 Netzwerke

4.1 Denial of Service (DoS) Attacken

4.1.1 Typen

Die heimtückischste Form von Denial of Service Attacken sind Bandbreiten-Konsum Attacken. Ein Angreifer wird versuchen, sämtliche vorhandene Bandbreite zu einem bestimmten Netzwerk aufzubrauchen. Dies kann einerseits dadurch erfolgen, dass der Angreifer eine schnellere Netzwerkverbindung als sein Opfer hat, andererseits dadurch, dass er andere Rechner verwendet, um sein Opfer zu überfluten.

Eine zweite Form von Denial of Service Attacken ist jene, welche im Gegensatz zu Netzwerk-Ressourcen sämtliche System-Ressourcen aufbraucht. Dies umfasst CPU-Verwendung, Memory, Dateisystem oder andere System-Prozesse. “Resource starvation” DoS Attacken haben führen meistens zu einem System-Absturz, zu einem vollen Dateisystem oder zu abgestürzten Prozessen.

Die dritte Art von DoS Attacken nutzen Fehler von Applikationen, Betriebssystemen oder eingebetteten Logik-Chips aus, normalerweise nicht korrektes Exception-Handling. Vielfach sendet ein Angreifer nicht-RFC-konforme Pakete um festzustellen, ob der Netzwerk-Stack eine Ausnahme korrekt behandelt, oder ob das Paket zu einem “kernel panic” oder einem Systemabsturz führt. Als Beipiel eines eingebetteten Systems ist die Pentium f00f DoS Attacke bekannt, welche einem User-Prozess erlaubte, jedes Betriebssystem mit Ausführung der Instruktion 0xf00fc7c8 zum Absturz zu bringen.

4.1.2 Generische DoS-Attacken

Smurf

Die Smurf-Attacke ist eine der meistgefürchteten DoS Attacken aufgrund des starken Verstärkungseffekts der Attacke. Der Verstärkungseffekt wird dadurch erreicht, dass ein gerichtetes broadcast ping an ein Netzwerk von Systemen gesendet wird, die auf solche Requests antworten werden.

Ein Angreifer sendet ein gespooftes ICMP ECHO Paket an die Broadcast Adresse des verstärkenden Netzwerks. Die Absender-Adresse wird so gefälscht, als ob das System des Opfers entsprechenden Request gesendet habe. Da das ECHO Paket an die Broadcast Adresse gesendet wurde, werden alle Systeme des verstärkenden Netzwerks dem Opfer antworten. Sendet ein Angreifer nur 14KB anhaltende ICMP Daten und besteht das verstärkende Netzwerk aus 100 Systemen mit einer schnellen Netzwerkverbindung, so bewirkt dies 14Mbps und legt selbst ein Netzwerk mit T1-Anbindung (1.544Mbps) lahm.

Um zu vermeiden, als verstärkendes Netzwerk missbraucht zu werden, sollte die gerichtete Broadcast Funktionalität auf dem äussersten Router eines

Unternehmensnetzwerks ausgeschaltet werden (z.B. `no ip directed-broadcast` Befehl für Cisco Router).

SYN Flood

Wenn eine TCP-Verbindung hergestellt wird, geschieht dies mittels eines Dreiwege-Handshake. Die die Verbindung initiierende Maschine A sendet ein SYN-Paket an einen spezifischen Port auf Maschine B, die sich im Zustand LISTEN befindet. Nach dem SYN-Paket befindet sich die potentielle Verbindung auf System B im Zustand SYN_RECV. System B versucht nun, ein SYN/ACK-Paket an A zurückzusenden. Wenn alles richtig verläuft, sendet A ein ACK-Paket zurück und die Verbindung gelangt in den ESTABLISHED Zustand.

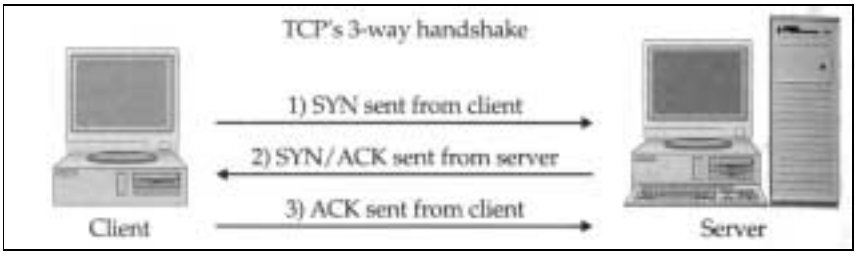


Abbildung 4: TCP Dreiwege-Handshake

Das Verfahren hat aber einige gewichtige Nachteile, die bei einer DoS-Attacke ausgenutzt werden. Die meisten Systeme allokieren bei Erhalt eines SYN-Pakets eine gewisse Menge an Ressourcen für die potentielle Verbindung. Obwohl die Systeme hunderte von Verbindungen pro Port gleichzeitig offen haben können, ist die Menge der möglichen Requests an potentiellen Verbindungen begrenzt. Es braucht nur etwa ein Dutzend potentielle Verbindungen um sämtliche Ressourcen zu allokieren, die zum Aufbau einer Verbindung vorhanden sind. Genau diesen Mechanismus verwendet ein Angreifer bei einer SYN Flood Attacke.

Der Angreifer sendet SYN-Pakete von Maschine A an Maschine B. Jedoch werden die Pakete gespooft, d.h. der Angreifer versieht die Pakete mit einer nicht existierenden Absender-Adresse. Das Zielsystem wird nun versuchen ein SYN/ACK-Paket an die gespoofte Adresse zu senden. Existiert das gespoofte System, würde es normalerweise mit einem RST-Paket an B antworten, da es ja nie versucht hat eine Verbindung zu initiieren. Der Angreifer hat jedoch ein unerreichbares System ausgewählt, B wird also nie ein RST-Paket erhalten. Die potentielle Verbindung bleibt im SYN_RECV Zustand und in einer Verbindungswarteschlange. Die potentielle Verbindung wird erst wieder gelöscht, sobald der connection-establishment Timer abgelaufen ist – dies kann je nach IP-Implementation zwischen 75 Sekunden bis 23 Minuten dauern. Weil die Verbindungswarteschlange sehr klein ist braucht der Angreifer nur ein paar SYN-Pakete alle 10 Sekunden zu senden um einen Port vollständig lahmzulegen. Mittels SYN Flood Attacken ist es also möglich, mittels geringer Bandbreite ganze Systeme lahmzulegen. Es handelt sich um eine Stealth-Attacke, da die Pakete gespooft sind und es somit sehr schwierig ist, den Angreifer zu identifizieren.

Mögliche Gegenmassnahmen:

- Vergrössern der Verbindungswarteschlange: Dies ist keine optimale Lösung, da dazu zusätzliche Systemressourcen verbraucht werden, was die Systemleistung möglicherweise einschränkt.
- Verringern der Connection Establishment Timeout Periode: Dies ist auch keine optimale Lösung, da der Angreifer nur ein bisschen mehr SYN-Pakete senden muss.
- Einsetzen von Intrusion Detection Systems (IDS): Einige Systeme können SYN-Attacken erkennen und dagegen aktiv reagieren. Ein SYN-Angriff wird am Fluss von SYN-Paketen ohne folgende Antworten erkannt. Ein IDS sendet nun dem unter Attacke befindlichen System RST-Pakete, die mit den gespooften SYN Requests übereinstimmen. Das sich unter Attacke befindende System kann so seine Warteschlange leeren.

4.1.3 Verteilte DoS-Attacken (DDoS, Distributed Denial of Service Attacks)

Verteilte DoS-Attacken funktionieren dermassen, dass ein Angreifer versucht, mit möglichst vielen anderen Systemen gemeinsam ein Zielsystem lahmzulegen oder die gesamte Netzwerkbandbreite aufzubrauchen. Der erste Schritt besteht darin, auf möglichst vielen Systemen administrativen Zugriff zu erlangen. Dies geschieht mittels Scripts, die ständig nach schlecht konfigurierten und verwundbaren Rechnern mit Sicherheitslücken suchen. Sobald Zugriff zu einem System erlangt ist, wird der Angreifer DDoS-Software installieren und ausführen. Die meisten DDoS-Server (bzw. Daemons) warten nach Ausführung auf weitere Instruktionen bis sie ein Zielsystem angreifen. Der Angreifer kann also auf den richtigen Zeitpunkt warten, bis er den Befehl für eine Attacke erteilt.

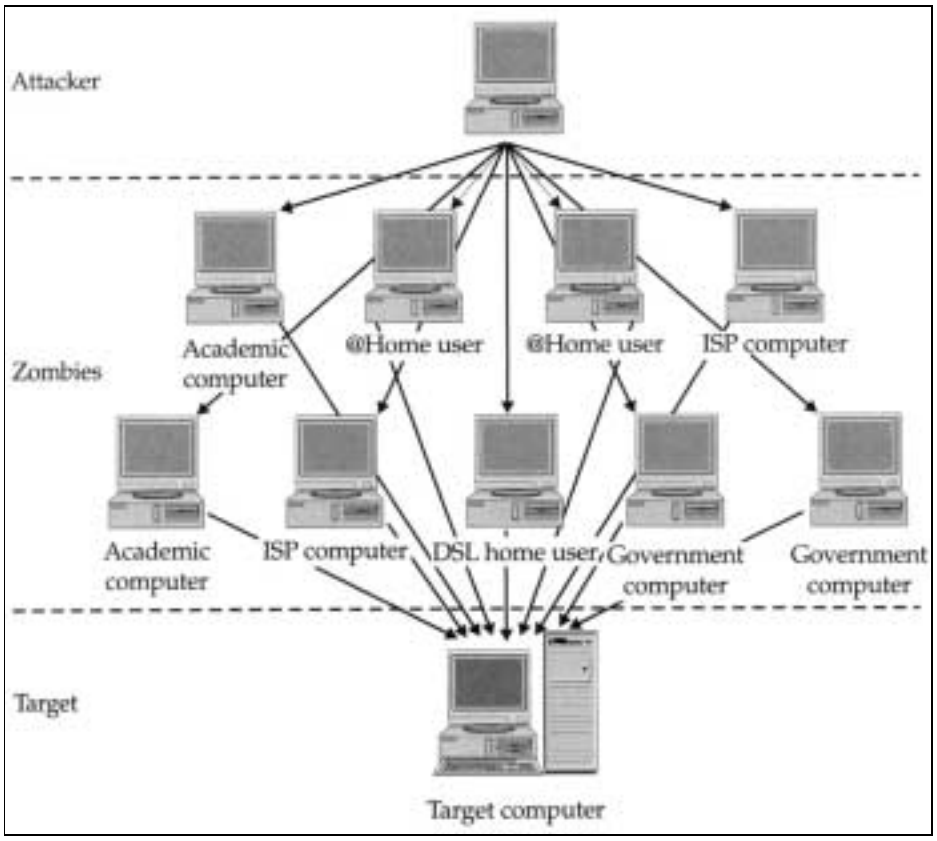


Abbildung 5: Ablauf einer DDoS-Attacke

Tribe Flood Network (TFN)

TFN war das erste öffentlich verfügbare UNIX-basierende DDoS-Tool. TFN besitzt eine Client- und eine Server-Komponente: Der Angreifer installiert die Server-Komponente auf entfernten, gehackten Systemen und kann später mittels einem einzigen Befehl auf dem Client einen Angriff starten. Mögliche Attaken sind ICMP Flood, Smurf, UDP und SYN Floods.

Die beste Abwehr gegen solche Attacken ist sich davor zu schützen, gehackt zu werden. D.h. unnötige Systemdienste abschalten, die aktuellen Betriebssystem- und Anwendungspatches installieren und Datei- und Verzeichnisberechtigungen richtig setzen. Da die TFN Kommunikation über ICMP erfolgt, ist es auch möglich sämtlichen eingehenden ICMP-Verkehr zu blockieren.

Neben Tribe Flood Network existieren eine Menge weiterer DDoS-Attacken wie Trinoo, Stacheldraht, TFN2K, WinTrinoo usw.

5 Software

Die gefürchtetste und häufigste Attacke bei Software-Produkten ist der sog. *Buffer Overflow*. Er ist besonders deshalb sehr gefährlich, weil ein Angreifer unter Umständen beliebigen Code auf der Zielmaschine ausführen kann.

5.1 Buffer Overflow

Für einen Buffer-Overflow ist letztlich immer der Programmierer der Anwendung oder des Systems verantwortlich. Jedes Programm legt zur Laufzeit lokale Variablen, Übergabeparameter für Funktionen sowie Rücksprungadressen für Unterprogramme im Arbeitsspeicher ab. Dieser spezielle Bereich wird als Stack bezeichnet und ist durch das Betriebssystem nicht vor ungewollten Änderungen geschützt. Schreibt das Programm beispielsweise eine zu lange Zeichenkette in eine lokale Puffervariable, überschreibt es dabei die darauf folgenden Variablen und unter Umständen auch die Rücksprungadresse - es kommt zu einem Buffer-Overflow.

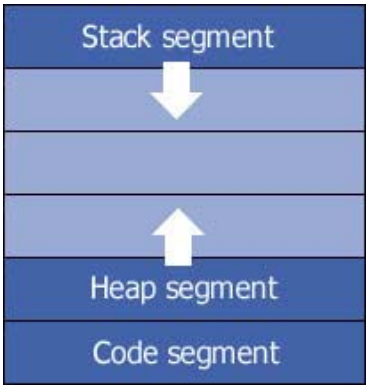
Besonders häufig tritt dieser Fall bei Programmen auf, die in der Programmiersprache C geschrieben sind. Sie bietet Funktionen wie `strcpy()` für das Kopieren eines Strings, die nicht kontrollieren, ob der Speicherbereich an der Zieladresse groß genug für die Daten ist. Ist dieser Puffer zu klein, überschreibt die Kopierfunktion gnadenlos die folgenden Datenfelder. Viele Programmierer versäumen immer noch, stattdessen die Funktion `strncpy()` zu verwenden, bei der man die Zahl der zu schreibenden Zeichen begrenzen kann.

In den meisten Fällen führt ein Buffer-Overflow zum Absturz des betroffenen Programms. Entweder, weil Variablen mit unsinnigen Werten überschrieben wurden, sodass das Programm nicht wie erwartet funktioniert, oder weil die Rücksprungadresse nicht mehr stimmt. Landet an deren Stelle ein quasi zufälliger Wert, springt das Programm nach dem Beenden der Funktion an eine zufällige Speicheradresse und meldet einen ‘Speicherzugriffsfehler’.

Viel schlimmer ist es jedoch, wenn diese Rücksprungadresse auf speziell präparierte Speicherbereiche verweist, die ‘sinnvollen’ Code enthalten. Gelingt es einem Angreifer - zum Beispiel über einen Eingabestring - eigene Anweisungen auf dem Stack abzulegen, kann er durch gezielte Manipulation der Rücksprungadresse dafür sorgen, dass dieser angesprungen und ausgeführt wird: Er hat einen ‘Exploit’ entwickelt.

Zum Verständnis, wie solche Exploits funktionieren, ist ein kleiner Exkurs in die Speicherverwaltung notwendig. Die nachfolgenden Beispiele beziehen sich auf die i386-Architektur, lassen sich aber prinzipiell auch auf andere Prozessorarchitekturen übertragen.

Der Prozessor ist eine Recheneinheit, ergänzt um Register. Die Register kann man sich als interne Speicherstellen vorstellen, welche über symbolische Namen (EAX, EIP, ...) angesprochen werden. Die meisten dienen als allgemeine Zwischenspeicher



für Rechenoperationen, Parameterübergabe und so weiter. Doch einige dieser Register erfüllen spezielle Aufgaben. Dazu gehören insbesondere der Instruction Pointer (EIP), der Base Pointer (EBP) und der Stack Pointer (ESP). Das vorangestellte 'E' steht für Extended und unterscheidet die 32-Bit-Register von ihren 16 Bit großen Pendants.

Der Instruction Pointer zeigt immer auf die Adresse des nächsten auszuführenden Maschinenbefehls. Nachdem der Prozessor eine Anweisung abgearbeitet hat, lädt er den nächsten Befehl von dieser Adresse und setzt den Zeiger weiter. Beim Verzweigen in eine Unterfunktion überschreibt das Programm den Instruction Pointer mit deren Startadresse. Base Pointer (EBP) und Stack Pointer (ESP) beschreiben den lokalen Speicherbereich einer Unterfunktion.

Bei modernen Betriebssystemen arbeitet jedes Programm in einem eigenen, virtuellen Adressraum, dessen Adressen die Hardware - konkret die Memory Management Unit (MMU) - erst bei Bedarf 'echten' Speicher zuordnet. In diesem legt das Betriebssystem beim Start eines Programms drei so genannte Segmente an: das Code-Segment, das Daten-Segment (auch Heap genannt) und das Stack-Segment. 'Unten' im Adressraum - also an den niedrigen Adressen - befindet sich das Code-Segment mit den eigentlichen Maschinenbefehlen des Programms. Es ist in Größe und Inhalt fest und zumeist gegen Überschreiben geschützt. Das heißt, ein Schreibversuch auf diese Adresse produziert eine Speicherschutzverletzung ('segmentation fault').

Globale Daten und Konstanten legt das System im Daten-Segment ab, das direkt 'über' dem Code-Segment liegt. Dort reserviert es auch Platz für Speicherbereiche, die das Programm zur Laufzeit mit dem Systemaufruf *malloc()* anfordert. Deshalb kann der Heap zur Laufzeit nach 'oben' wachsen.
Das Stack-Segment ist ein Zwischenspeicher für lokale Variable und gesicherte Prozessorregister, die das Programm zu einem späteren Zeitpunkt wieder benötigt. Der Stack beginnt am oberen Ende des Adressraums und wächst nach 'unten'. Er funktioniert als Last-In-First-Out-Puffer, ähnlich wie ein Stapel, den man erst abräumen muss, bevor man an früher abgelegte Daten kommt.

Die beiden Basisoperationen für den Stack sind 'push' und 'pop'.

```
push %eax
```

sichert den Inhalt des Prozessorregisters auf dem Stack, zum Beispiel bevor das Programm in eine Unterfunktion verzweigt, die deren Inhalt verändern könnte. Nach der Rückkehr holt es den gespeicherten Wert mit *pop %eax* wieder zurück. Dabei zeigt der Stack-Pointer (ESP) immer auf das aktuelle Ende des Stacks.

C-Programme legen die Übergabeparameter einer Funktion, die Rücksprungadresse und lokale Variable auf dem Stack ab. Auf die lokalen Variablen kann die Funktion über einen Offset zum so genannten Base Pointer zugreifen (EBP), der auf den Anfang des Datenbereichs einer Funktion zeigt (Stackframe).

Untenstehende Abbildung zeigt ein einfaches Beispielprogramm in C, Auszüge aus dem zugehörigen Assembler-Code und den Stack-Inhalt während der Ausführung der Funktion.

C-Code

```
void function(int a, int b, int c) {
     char buffer1[8];
     char buffer2[16];
     ...
}

void main() {
     function(1,2,3);
}
```

Assembler-Code
(Auszug aus "gcc -S -o example1.s example1.c")

```
function:
    pushl %ebp           # sichert EBP
    movl %esp,%ebp       # kopiert ESP nach EBP
    subl $24,%esp        # schafft Platz f. buffer1+2
    movl %ebp,%esp       # korrigiert EBP
    ...
    popl %ebp
    ret

main:
    pushl %ebp
    movl %esp,%ebp
    pushl $3            # Parameter auf den Stack
    pushl $2
    pushl $1
    call function       # Funktionsaufruf
    addl $12,%esp       # Stack aufräumen
```

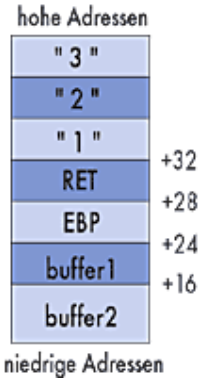


Abbildung 6: Bei jedem Funktionsaufruf landen Parameter, Rücksprungadresse und lokale Variablen auf dem Stack.

Auf dem Stack befinden sich unter anderem die lokalen Puffer *buffer1* und *buffer2* und die gespeicherte Rücksprungadresse. Kopiert man in der Funktion mit

```
strcpy(buffer1, buffer2);
```

den Inhalt des größeren, zweiten Puffers in den ersten, überschreibt diese Operation auch diese Rücksprungadresse. Der abschließende Assemblerbefehl 'ret' holt diesen quasi zufälligen Wert vom Stack und schreibt ihn in den Instruction Pointer. Im nächsten Arbeitsschritt versucht der Prozessor, von dieser Adresse den nächsten Befehl zu laden - was in der Regel fehlschlägt und eine Speicherschutzverletzung erzeugt.

Dass das nicht immer so sein muss, demonstriert untenstehendes Beispiellisting. Hier erhöht das Programm den Wert der Rücksprungadresse um 8 – mit dem Resultat, dass es direkt den printf-Aufruf anspringt. Der Befehl ‘x=1’ kommt nicht zur Ausführung. Probieren Sie es aus: das Programm gibt ‘0’ aus.
Um in ein System einzubrechen, wollen die Angreifer jedoch eigenen Code ausführen. Dieser gelangt zumeist über lange Eingabestrings auf den Stack, die neben der neuen Sprungadresse auch die Maschinenbefehle enthalten. Dieser eigene Code wird auch als Payload bezeichnet und ist quasi das ‘Herz’ eines Buffer-Overflow-Exploits.

```
1 void function(int a, int b, int c) {
2    char buffer1[8];
3    char buffer2[16];
4    int *ret;
5
6    ret = buffer1 + 12;
7    (*ret) += 8;
8 }
9
10 void main() {
11   int x;
12
13   x = 0;
14   function(1,2,3);
15   x = 1;
16   printf("%d\n",x);
17 }
```

Abbildung 7: Die Funktion überschreibt ihre eigene Rücksprungadresse, und das Programm gibt ‘0’ aus.

Beim Erstellen eines Exploits treten noch diverse Probleme auf, die sich jedoch in der Regel alle lösen lassen. Es beginnt mit der Adresse, an der sich der Code befindet. Sie muss als absoluter Wert an die Stelle der Rücksprungadresse geschrieben werden. Da jedoch die absolute Position der lokalen Variablen auf dem Stack nicht fest ist, weiß der Angreifer nicht genau, wo sein Code beginnt. Die genaue Adresse hängt unter anderem von der Anzahl und Länge der Umgebungsvariablen ab, die am Beginn des Stacks liegen. Als Workaround kommt das so genannte NOP-Sliding zum Einsatz. Dabei schreibt man vor den eigentlichen Code eine Reihe von No-Operation-Befehlen (NOPs). Landet der Sprung irgendwo in diesem NOP-Bereich, arbeitet die CPU die übrigen NOPs ab und kommt danach zum eigentlichen Exploit-Code.

Das nächste Problem hat damit zu tun, dass Strings in C immer mit einer ‘0’ beendet werden. Deshalb interpretiert das Programm das erste Auftreten des Werts 0x00 als das Ende des Eingabe-Strings. Dieser darf also nicht in der Payload vorkommen. Hier kann man zu Tricks greifen, und Zuweisungen wie ‘`mov 0,eax`’ durch das gleichwertige ‘`xor eax,eax`’ ersetzen. Eleganter ist es jedoch, den gesamten Code über die XOR-Verknüpfung mit einer Zahl Y zu kodieren. Diese darf dann jedoch nicht selbst im Maschinen-Code vorkommen, da dies wiederum zu einem 0x00 führen würde. Die Dekodierung erfolgt dann am Beginn des Programms mit wenigen Zeilen Assembler. Ähnliche Sonderbehandlung muss man unter Umständen den Zeichen für Tabulator und Linefeed angedeihen lassen.

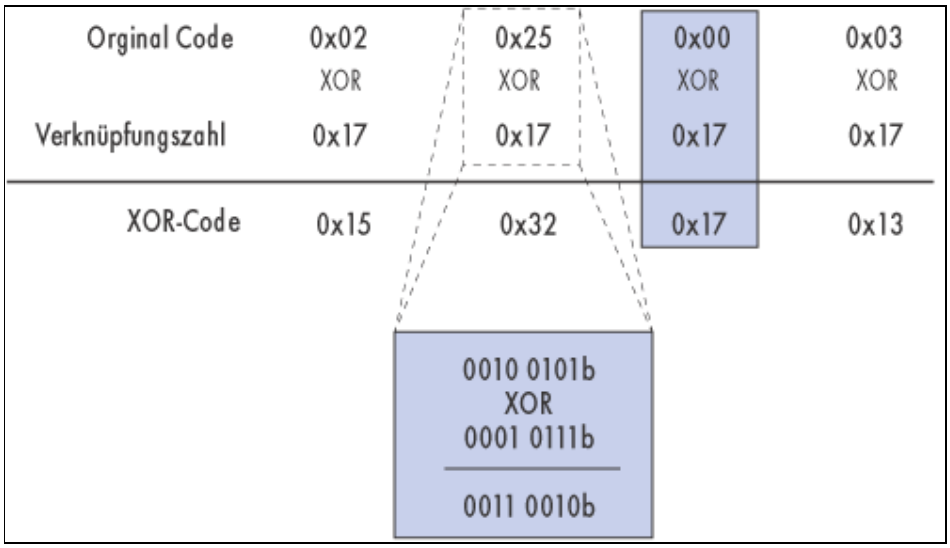


Abbildung 8: Die XOR-Verknüpfung mit 0x17 entfernt den Wert 0x00 aus dem String (Quelle: c't 23/2001)

Schließlich muss man auch in einem solchen Exploit gelegentlich auf eigene Daten wie den String '/bin/bash' zugreifen.

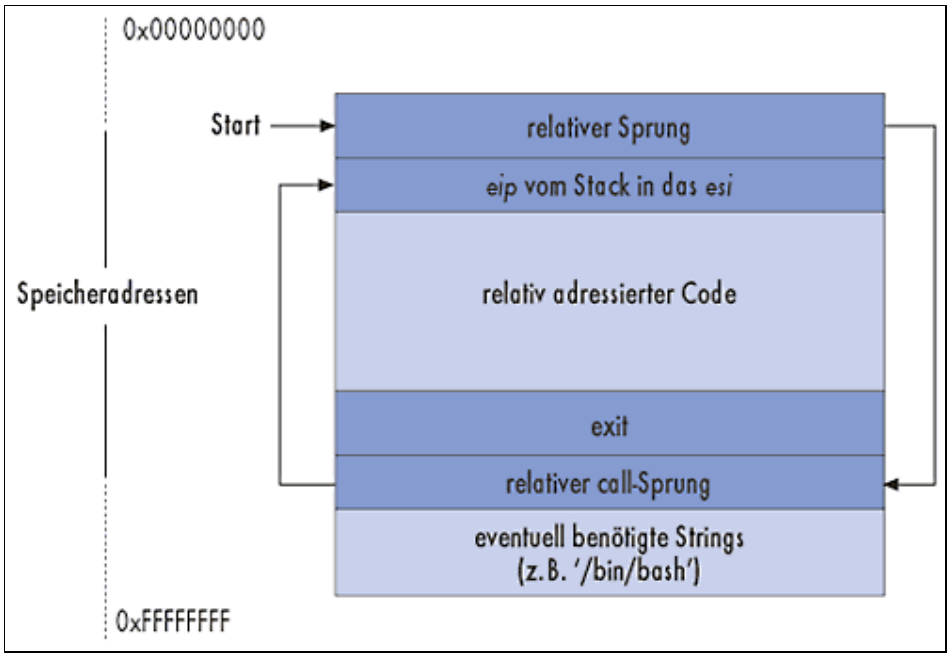


Abbildung 9: Der Aufbau eines ‚Strings' für einen Buffer-Overflow-Exploit. (Quelle: c't 23/2001)

Da die absolute Position des Strings im Speicher nicht bekannt ist, muss die Adressierung relativ erfolgen. Doch relativ wozu? Auch hier greifen die Programmierer von Exploits zu einem Trick: Sie springen mit einem relativen Jump-Befehl an das Ende ihres Codes, hinter dem sich die benötigten Variablen befinden.

Dort simulieren sie via ‘call’ einen Funktionsaufruf auf den nächsten abzuarbeitenden Befehl. Dabei schiebt das System die nächste Adresse - also die des Stringanfangs - auf den Stack. Von dort kann man sie via ‘pop’ in ein beliebiges Register befördern. Schon hat der Programmierer seinen Zeiger auf den eigenen Datenbereich.
Damit steht dem eigentlichen Exploit nichts mehr im Weg. Doch natürlich will der Angreifer dabei Dateizugriffe oder den Start eines Programms nicht selbst in Assembler programmieren. Deshalb greift er auf Funktionen des jeweiligen Betriebssystems zurück. Linux bietet über den Software-Interrupt 0x80 Zugang zu allen wichtigen Funktionen - alte Hasen kennen dieses Verfahren noch vom DOS-Interrupt 0x21. Um eine neue Datei anzulegen, genügt es, vor dem Aufruf von ‘int 0x80’ einige Register entsprechend zu präparieren. EBX muss die Adresse des Strings mit dem Dateinamen enthalten, ECX sorgt für passende Zugriffsrechte und EAX wählt mit dem Wert 0x8 den Systemaufruf create() aus. Mit 0x80 erhält man Zugriff auf die Funktion execve(), über das man externe Programme wie die Shell ‘/bin/sh’ starten kann.

Unter Windows kann der Exploit-Code direkt alle Funktionen des Windows-APIs nutzen, die das Programm einbindet. Befindet sich darunter die Funktion LoadLibrary, kann der Angreifer auch beliebige Funktionen nachladen. Beim Aufruf der Windows-API-Funktionen müssen sich die Übergabeparameter wie bei einem normalen Funktionsaufruf in der richtigen Reihenfolge auf dem Stack befinden.

Der beste Schutz gegen Buffer-Overflows ist sicherheitsbewusste Programmierung. Software-Entwickler sollten sich unbedingt über die entsprechenden Fallstricke informieren und eine bewusst defensive Programmierung anstreben. Ganz besonders gilt das für die Entwickler von Programmen mit Netzwerkfunktionen, da durch diese auch Angriffe übers Netz möglich sind. Leider ist es mit dem auch hier zitierten strcpy() längst nicht getan - auch viele andere C-Funktionen bergen die Gefahr eines unerwarteten Pufferüberlaufs.

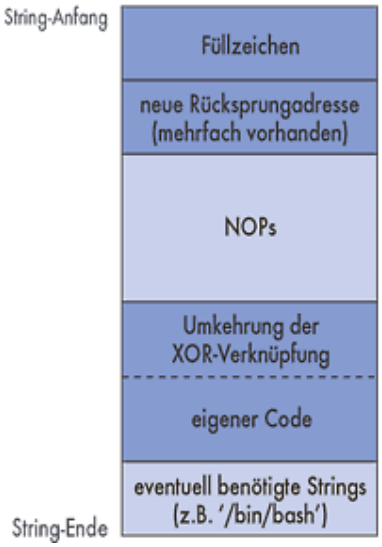


Abbildung 10: Über einen relativen Sprung beschaffen sich die Entwickler eines Exploits die Adressen der eigenen Daten.

Fein raus ist, wer Java statt C beziehungsweise C++ verwenden kann. Die Java-Plattform überwacht zur Laufzeit die Grenzen der Speicherbereiche und nimmt damit dem Entwickler eine große Last ab. Aber auch mit C/C++ kann man auf diverse Hilfsmittel zurückgreifen, die zumindest helfen, solche Sicherheitslücken zu vermeiden.

StackShield sichert in Linux-Programmen bei jedem Funktionsaufruf die Return-Adresse und korrigiert sie bei Bedarf vor dem Rücksprung. Das Programm fügt dazu entsprechenden Code am Beginn und Ende jedes Funktionsaufrufs ein.

Auch StackGuard versucht, auf Unix-Systemen die Rücksprungadresse bei Funktionsaufrufen zu schützen. Dazu platziert es auf dem Stack direkt daneben ein so genanntes Canary. Dahinter verbirgt sich ein spezielles Kontrollzeichen, dessen Wert StackGuard vor jedem Rücksprung aus einer Funktion überprüft. Hat er sich geändert, schreibt das Programm eine Warnmeldung ins Syslog und beendet sich. Stackguard ist als Compiler-Patch zu gcc realisiert.
Sowohl bei StackShield als auch bei StackGuard muss man den Quelltext des Programms neu übersetzen - sofern man Zugang dazu hat. Ist das nicht der Fall, hilft die Funktionsbibliothek libsafe weiter. Sie ersetzt gefährliche Bibliotheksaufrufe durch eigene Versionen, die zusätzlich zur eigentlichen Funktion das Überschreiben des eigenen Stackframes verhindert. Damit können zwar immer noch Pufferüberläufe vorkommen; da aber die Rücksprungadresse außerhalb des Stack-Frames liegt, lässt sich der Kontrollfluss des Programms nicht mehr manipulieren. libsafe ist unter Linux als dynamische Bibliothek realisiert, die sich zwischen das Programm und die eigentlichen System-Bibliotheken schaltet.

Um das Problem an der Wurzel zu packen, könnte man den Stack als nicht ausführbar markieren. Damit löst der Versuch, Code auf dem Stack auszuführen, eine Speicherschutzverletzung aus. Dies muss allerdings auf Betriebssystemebene implementiert sein und bringt diverse Kompatibilitätsprobleme mit sich. Von Solar Designer gibt es einen entsprechenden Patch für Linux. Allerdings haben Sicherheitsexperten bereits darauf hingewiesen, dass auch dies nicht der Weisheit letzter Schluss ist. So lassen sich Systemaufrufe wie system() auch direkt über die so genannte Procedure Linkage Table (PLT) anspringen.

Auch das PaX-Team hat einen Linux-Patch entwickelt, der sich spezielle Fähigkeiten der Intel-Hardware zu Nutze macht, um Daten-und Stack-Segment als nicht ausführbar zu markieren. Außerdem gibt es von SecureWave mit SecureStack eine Implementierung für Windows NT/2000, bei der allerdings derzeit die Windows-2000-Version noch übermäßige Performanceeinbußen mit sich bringt.

Gesicherte Betriebssysteme

Ein ganz anderes Konzept verfolgen gesicherte Betriebssysteme wie Argus Pitbull oder das Security-Enhanced Linux der NSA. Diese unternehmen gar nicht erst den Versuch, Buffer-Overflows zu verhindern, sondern versuchen deren Konsequenzen durch sehr detaillierte Rechtevergabe unter Kontrolle zu halten. So hat ein Angreifer selbst mit einer Root-Shell, die er durch einen Buffer-Overflow im Web-Server erlangt hat, nur sehr eingeschränkte Zugriffsmöglichkeiten auf das System. Dass sich auch solche Systeme hacken lassen, zeigte ein erfolgreicher Einbruch bei einem Hacking-Contest von Argus, bei dem die Angreifer ein Sicherheitsproblem des eingesetzten Solaris-Kernel ausnutzten.

Buffer-Overflows werden sicher auch in den nächsten Jahren eines der zentralen Sicherheitsprobleme darstellen. Neben den Stack-orientierten Exploits dürften in nächster Zeit auch vermehrt Exploits auftauchen, die sich Pufferüberlaufe in statischen Variablen auf dem Heap zu Nutze machen. Erste Veröffentlichungen dazu sind bereits im Internet erschienen. Solange Entwickler unter Zeitdruck immer neue

Funktionen und Features in Programme einbauen und Anwender sich mit regelmäßigen Patches für das Sicherheitsloch des Monats zufrieden stellen lassen, wird sich an dieser Situation auch nichts ändern.

6 Wireless LAN

Die Sicherheit vieler heutiger Funknetzwerke basiert auf dem *WEP-Algorithmus* (Wired Equivalent Privacy), welcher Teil des 802.11 Standards ist. Er garantiert die Verschlüsselung der Daten und Authentifizierung der Benutzer in Funknetzwerken und verhindert, dass ein unberechtigter Benutzer mit einer Wireless LAN-Karte den Datenverkehr mithören kann. Auch hier wurden vor einiger Zeit grosse Sicherheitslücken entdeckt, die sich im Zusammenhang mit der Verwendung von WEP ergeben.

WEP baut auf einem geheimen Schlüssel auf, der zwischen der Mobilstation (z.B. Laptop) und einem Access Point geteilt wird. Der Schlüssel wird einerseits verwendet, um die Pakete zu verschlüsseln, andererseits wird auch eine Integritätsprüfung durchgeführt, um sicherzustellen, dass der Inhalt der Nachricht nicht verändert wurde.

WEP verwendet den RC4-Verschlüsslungsalgorithmus, der als Stream Cipher bekannt ist. Eine Stream Cipher expandiert einen kurzen Schlüssel in einen unendlichen pseudo-zufälligen Key Stream. Der Sender führt nun eine XOR-Operation mit den zu verschlüsselnden Daten und dem Key Stream aus. Der Empfänger kennt den gleichen Key und somit denselben Key Stream und wendet die XOR-Operation erneut auf die verschlüsselten Daten an und erhält das Original wieder.

Diese Betriebsart macht Stream Ciphers verwundbar gegen verschiedene Attacken. Kehrt ein Angreifer ein Bit in den verschlüsselten Daten (Ciphertext), so wird nach dem Entschlüsseln das zugehörige Bit in den unverschlüsselten Daten (Klartext) gekehrt. Kann ein Angreifer zwei verschlüsselte Pakete auffangen, die mit derselben Stream Cipher verschlüsselt wurden, ist es möglich das XOR der beiden Klartexts zu erhalten. Kennt man die XOR-Verknüpfung der unverschlüsselten Pakete, ist es unter Umständen möglich mittels statistischen Attacken auf den Klartext zu schliessen. IP Daten sind häufig sehr vorhersagbar und umfassen eine Menge Redundanz. Diese Redundanz kann verwendet werden, um die Anzahl der Möglichkeiten der Inhalte der Nachrichten einzuschränken. Die statistische Attacke wird umso leichter, je mehr Daten mit derselben Stream Cipher verschlüsselt wurden. Ist einmal ein Klartext bekannt so ist die Entschlüsselung aller anderen Pakete trivial.

WEP hat Gegenmassnahmen gegen diese beiden Formen von Angriffen: Um sicherzustellen, dass ein Paket nicht während dem Transport geändert wurde, benutzt es ein Integrity Check (IC) Feld im Paket. Um zu vermeiden, dass zwei Ciphertexts mit dem gleichen Schlüsselstrom verschlüsselt werden, wird ein Initialisierungsvektor (IV) verwendet, der den geheimen geteilten Schlüssel

vergrössert und für jedes Paket einen verschiedenen RC4-Schlüssel produziert. Der IV ist auch im Paket enthalten. Diese beiden Massnahmen wurden allerdings schlecht implementiert, deshalb ist das Verfahren nicht sicher.

Das Integrity Check Field wurde als CRC-32 Prüfsumme implementiert, die ein Teil der verschlüsselten Payload des Pakets ist. Jedoch ist CRC-32 *linear*, das bedeutet, dass es möglich ist, die Bitdifferenz von zwei CRCs aufgrund der Bitdifferenz der Pakete zu berechnen, über deren sie genommen wurde. In anderen Worten bewirkt das Umkehren von n Bits in der Nachricht einen deterministischen Satz Bits in der CRC, die geändert werden müssen, um eine korrekte Prüfsumme der geänderten Nachricht zu produzieren. Da das Umkehren der Bits nach einer RC4-Decodierung erfolgt, erlaubt dies dem Angreifer, willkürliche Bits in der verschlüsselten Nachricht umzukehren und die Prüfsumme richtig zu justieren, damit die resultierende Nachricht gültig erscheint.

Der Initialisierungsvektor in WEP ist ein 24-bit Feld, welches im Klartextteil einer Nachricht gesendet wird. Solch eine kleine Anzahl von Initialisierungsvektoren *garantiert* die Wiederverwendung des selben Key Streams. Ein ausgelasteter Access Point, der 1500 Byte grosse Pakete bei einer Datenrate von 11 Mbps sendet, erschöpft den Raum von Initialisierungsvektoren nach 1500*8/(11*10^6)*2^24 = ~18000 Sekunden oder 5 Stunden (die Zeitdauer kann sogar kleiner sein, da viele Pakete kleiner als 1500 Bytes sind). Deshalb gelingt es dam Angreifer recht schnell, zwei Ciphertexts zu sammeln, die mit dem gleichen Key Stream verschlüsselt wurden und eine statistische Attacke durchzuführen, um den Klartext zu erhalten. Noch schlechter ist es, wenn der gleiche Key von allen Mobilstationen verwendet wird – die Wahrscheinlichkeit einer IV-Kollision ist dann noch grösser. Z.B. setzt eine gewöhnliche Wireless LAN Karte von Lucent den IV immer auf 0 zurück wenn die Karte initialisiert wird und erhöht den IV für jedes Paket immer um 1. Zwei Karten, die ungefähr zur gleichen Zeit eingeschaltet werden, stellen einem Angreifer somit einen Überfluss an IV-Kollisionen zur Verfügung (noch schlimmer ist, dass die Änderung des IV-Vektors für jedes Paket nach Standard 802.11 optional ist!).

WEP enthält also grosse Sicherheitslücken. Mittels Mitschneiden von Paketen können unter Umständen schon nach relativ kurzer Zeit Daten entschlüsselt werden. Um sichere Kommunikation sicherzustellen ist eine Verschlüsselung auf einer höheren Protokollebene (z.B. IPsec[4]) unumgänglich.

7 Literaturverzeichnis

1. Scambray, J., McClure, S., Kurtz, G.; Hacking Exposed: Network Security Secrets & Solutions; Second Edition; 2001; Osborne/McGraw-Hill; ISBN: 0072127481
2. Silberschatz, A., Galvin, P. B.; Operating System Concepts; Fifth Edition; 1998; Addison-Wesley; ISBN: 0201542622

[4] IPsec (Internet Protocol Security) ist ein sich etablierender Standard für Sicherheit auf Netzwerk- oder Paketebene. IPsec ist nützlich für das Einrichten von VPN-Verbindungen (Virtual Private Networsk) und für Fernzugriff auf private Netzwerk via Modem.

3. c't: Magazin für Computertechnik (19/2001, 23/2001); Verlag Heinz Heise
4. Secure Programming for Linux and UNIX HOWTO; http://www.dwheeler.com/secure-programs/
5. Viega, J., McGraw, G.; Building Secure Software: How to Avoid Security Problems the Right Way; Addison Wesley Professional, ISBN: 0201721252X
6. StackShield; http://www.angelfire.com/sk/stackshield/
7. StackGuard; http://immunix.org
8. libsafe; http://www.avayalabs.com/project/libsafe/index.html
9. Probleme bei non-executable Stacks; http://www.insecure.org/sploits/non-executable.stack.problems.html
10. PaX; http://pageexec.virtualave.net
11. SecureStack; http://www.securewave.com/products/securestack/secure_stack.html
12. Smashing the Stack for Fun and Profit; http://www.phrack.org/show.php?p=49&a=14
13. Heap Overflows; http://www.w00w00.org/files/articles/
14. Mudge's tutorial on writing Buffer Overflows; http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html
15. Cert® Coordination Center; http://www.cert.org
16. Security of the WEP algorithm; http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html
17. IEEE 802.11 unsicher – WEP-Verschlüsselung wurde geknackt; http://www.golem.de/0108/15270.html