Überwachung von Netzwerken und Analyse aufgezeichneter Daten

Ronny Harbich
Stendal, den 11. Juni 2006

Vorwort

Ich danke hiermit Anke Szofer und Steffen Harbich für die lobenswerte Unterstützung bei dieser Arbeit. Besonders Anke Szofer ist es zu verdanken, dass Sprache und Grammatik deutlich verbessert wurden.

Für etwaige Kritik an ronny.harbich@student.uni-magdeburg.de wäre ich dem Leser sehr dankbar.

Inhaltsverzeichnis

1 Modell eines Netzwerks

Im Folgenden verwenden wir das in Abbildung 1 dargestellte Konzept eines Netzwerks. Wir werden also davon ausgehen, dass ein internes Netzwerk verwaltet wird, welches über ein *Gateway* Zugriff auf ein externes Netzwerk hat. Generell setzen wir eine Überwachung am Gateway an, da über ihm sämtliche Kommunikation mit dem externen Netzwerk stattfindet. Außerdem werden wir unsere Betrachtungen auf das *Internet Protocol (IP)* einschränken.

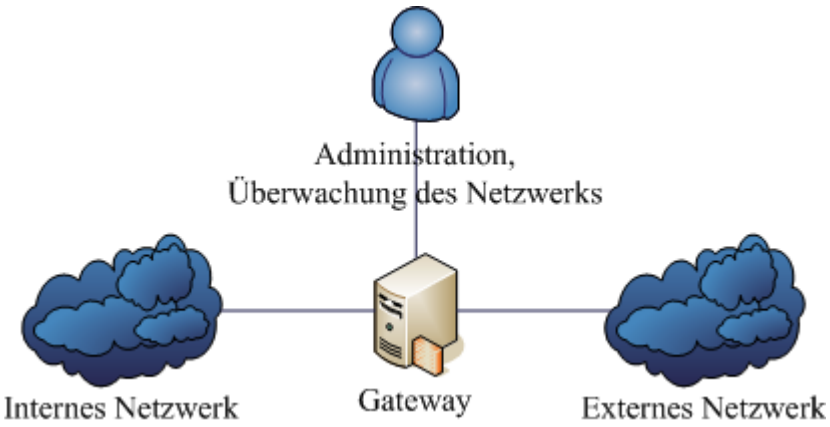

Abbildung 1: Konzept eines Netzwerks

2 Rechtfertigung einer Netzwerküberwachung

Die Administration eines Netzwerks sollte generell die Möglichkeit haben, bei Problemen Eingriffe ins Netzwerk vorzunehmen. Dies kann besonders dann der Fall sein, wenn das Netzwerk für illegale Aktivitäten verwendet wird. So können beispielsweise Schadprogramme wie *Würmer*, *Viren* und *trojanische Pferde* in das zu betreuende Netzwerk eingeführt oder vertrauliche Daten ausgeführt werden (Abbildung 2). Überdies kann auch eine unerwünschte Nutzung der durch das Netzwerk verbundenen Ressourcen von Dritten erfolgen. Um diese illegitimen Verwendungen zu entdecken, zu analysieren und ihnen dann vorzubeugen, bedarf es der weiteren Betrachtung technischer und rechtlicher Aspekte.

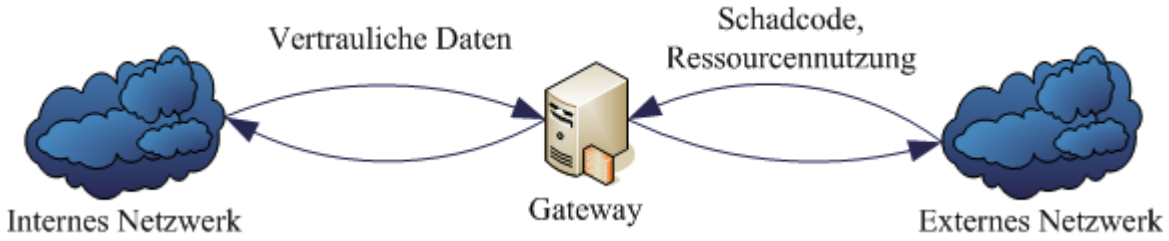

Abbildung 2: Bedrohungen eines Netzwerks

3 Theoretische und technische Möglichkeiten der Überwachung

3.1 Prävention illegitimer Nutzung

Bevor wir uns der Netzwerküberwachung zuwenden, diskutieren wir kurz denkbare Präventionsmaßnahmen illegitimer Benutzung. Ein gut organisiertes internes Netzwerk sollte a priori nur selektive Verbindungen nach außen hin zulassen. Dies kann etwa mit einer *Firewall* oder einem *Paketfilter* erreicht werden. Leider genügen diese Maßnahmen nicht zum vollständigen

Schutz vor den oben genannten Problemen, da es durchaus Möglichkeiten gibt, über die selektiv zugelassenen Verbindungen illegitime Handlungen durchzuführen.

3.2 Prinzipielle Schwierigkeiten der Überwachung

Zwei der größten Probleme bei einer Überwachung dürften wohl aufkommendes Datenvolumen und verschlüsselte Verbindungen sein. So wird ein Aufzeichnen der Inhalte einer Verbindung kaum realisierbar sein, da sie einfach zu viel Speicherplatz benötigen würden. Auch entfällt die Frage nach dem Protokollieren der Inhalte, sobald eine Verbindung verschlüsselt ist. Denn bei ordentlich implementierter Verschlüsselung existiert heute kein Weg, die codierten Daten in annehmbarer Zeit zu decodieren. In der Praxis angewandte und weit verbreitete Standards sind *Secure Shell (SSH)* und *Secure Sockets Layer (SSL)* etwa bei *HTTPS* (Abbildung 3). Der Administrator sollte den Nutzern nicht das Recht auf vertrauliche Übermittlung von Daten verweigern, wenngleich er aufgrund eventuell illegitimer Nutzung Kenntnis über den Inhalt der Verbindung haben sollte. In vielen Fällen wird nämlich ein Verschlüsseln der Daten unabdingbar sein, da beispielsweise Inhalte betrieblicher Geheimhaltung unterliegen.

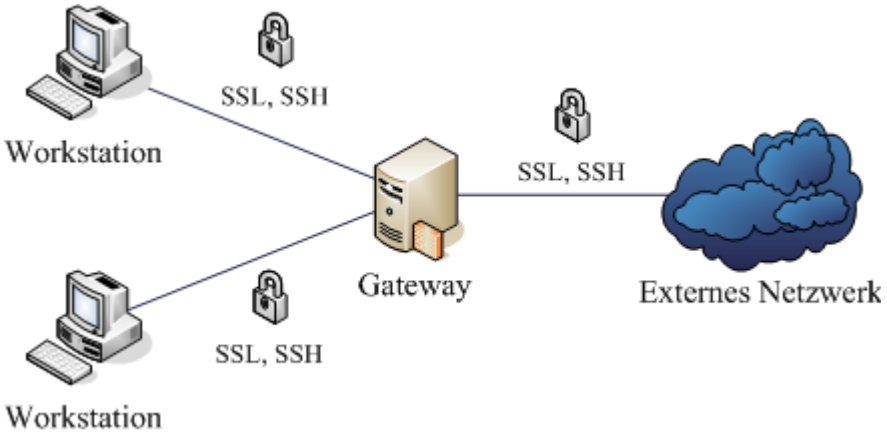

Abbildung 3: Verschlüsselte Kommunikation

Als andere Schwierigkeit der Netzwerküberwachung erweisen sich anonymisierte Verbindungen. Dies sind solche, bei denen die Inhalte nicht nur verschlüsselt, sondern Absender und Empfänger nicht ohne Weiteres bestimmbar sind. Eine derartige Anonymisierung kann realisiert werden, indem mehrere *Proxys* zwischen Absender und Empfänger geschaltet werden. Im Speziellen werden, statt einfacher Proxys, so genannte *Mixe* eingesetzt (Abbildung 4). Ein Benutzer verbindet sich nun mit einem ersten Mix, wobei die Inhalte der Verbindung verschlüsselt übertragen werden. Dieser erste verschlüsselt die Inhalte ein weiteres Mal und sendet sie zu einem zweiten Mix. Grundsätzlich können beliebig viele Mixe folgen, deren Betrieb von einander unabhängig ist. Erreichen die Inhalte schließlich den letzten Mix, so werden die Daten entschlüsselt und zu dem eigentlichen Empfänger weitergeleitet. Da an einem solchen System viele Benutzer beteiligt sind und jeder Mix die Verbindungsdaten der einzelnen Benutzer mischt, wird eine eindeutige Zuordnung der Datenströme sehr unwahrscheinlich. Somit haben jeweils die Mixe und der Empfänger nur Kenntnis vom jeweiligen Vorgänger, mithin weiß der Empfänger nur, dass ein Mix Inhalte gesendet hat. Dieses Anonymisierungsverfahren kann dann und nur dann manipuliert und überwacht werden, wenn jemand alle Mixe kontrolliert, sie also nicht mehr unabhängig sind. In der Praxis werden zur Zeit Programme wie *JAP* und *Tor* eingesetzt. Demnach haben wir keine realistische Aussicht, anonymisierte Verbindungen zurück zu verfolgen.

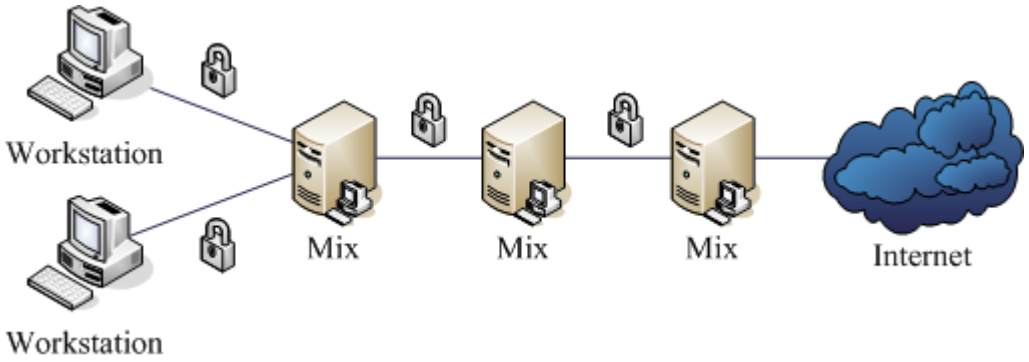

Abbildung 4: Prinzip der Anonymisierung

Ein größeres Problem stellt auch die Verwendung versteckter Datenübertragung dar. So besteht etwa die Möglichkeit (illegitime) Daten in andere (legitime) Daten zu verstecken. Dieses *Steganographie* genannte Prinzip findet häufig bei Bildern Anwendung. Dabei werden die zu versteckenden Informationen so ins Bild eingefügt, dass für einen Menschen die Unterschiede zwischen dem ursprünglichen und dem manipulierten Bild nicht nachvollziehbar sind. Eine weitere Methode Daten versteckt zu übertragen, ist das so genannte *Tunneln*. Hierbei werden die Daten eines Netzwerkprotokolls in ein anderes verlagert und verschlüsselt. Mit diesen Verfahren ist es ein Leichtes präventive Sicherheitsmaßnahmen wie Firewalls zu umgehen. Wird nun ein Netzwerk zur Übertragung versteckter illegitimer Daten missbraucht, gibt es generell kaum Wege dies zu erkennen.

3.3 Überwachung eines Netzwerks

Wie wir oben gesehen haben, ist eine Überwachung des Inhalts einer Übertragung praktisch kaum ausführbar. Mithin bleibt nur noch eine Überwachung der Metadaten einer Verbindung übrig. Wir können also Daten wie Quell- und Ziel-IP-Adresse, Länge des IP-Pakets und des IP-Paket-Kopfes, sowie *Ports* (Anschlussnummern) im Falle einer *TCP*- oder *UDP*-Verbindung protokollieren. Überdies besteht auch die einfache Möglichkeit, Datum und Uhrzeit einer Verbindung mit aufzuzeichnen.

Auf dem Markt existieren eine ganze Reihe von Programmen zur Protokollierung des Datenflusses im Netzwerk. Diese sind hauptsächlich *Sniffer*. Überdies gibt es so genannte *Intrusion Detection Systems (IDS)*, die statt nur die Daten aufzuzeichnen, sie gleichzeitig analysieren und auf eventuelle Störungen reagieren. Wir wollen hier solche Programme nicht weiter betrachten, sondern ein prinzipielles programmiertechnisches Vorgehen unter *Microsoft Windows 2000* und höher und *C# .NET 2.0* angeben. Um Verbindungen über das Internet Protocol zu überwachen, verwenden wir einfach einen entsprechenden *Socket* (System.Net.Sockets):

```
Socket MonitorSocket = new Socket(AddressFamily.InterNetwork,
     SocketType.Raw, ProtocolType.IP);
MonitorSocket.Bind(new IPEndPoint(ip, 0));
```

MonitorSocket ist hierbei ein Socket, der an die IP-Adresse *ip* gebunden wird und die Daten roh erhält. Diese rohen Daten beinhalten dann die oben genannten Informationen wie beispielsweise Quell- und Ziel-IP-Adresse. Natürlich können wir den Datenstrom nach gewünschten Informationen filtern und so nur die für uns wesentlichen Bestandteile speichern.

4 Rechtliche Rahmenbedingungen zur Überwachung

Wir weisen an dieser Stelle ausdrücklich darauf hin, dass Aktualität, Vollständigkeit und Richtigkeit der folgenden Aussagen nicht gewährleistet werden können.

Die Administration eines Netzwerks hat die in Deutschland geltenden Gesetze zur Überwachung einzuhalten. Besonders Artikel 10 des *Grundgesetzbuchs* räumt dem Deutschen Volk das *Fernmeldegeheimis* als eines ihrer grundlegenden Rechte ein:

(1) Das Briefgeheimnis sowie das Post- und Fernmeldegeheimnis sind unverletzlich.

(2) Beschränkungen dürfen nur auf Grund eines Gesetzes angeordnet werden. Dient die Beschränkung dem Schutze der freiheitlichen demokratischen Grundordnung oder des Bestandes oder der Sicherung des Bundes oder eines Landes, so kann das Gesetz bestimmen, daß sie dem Betroffenen nicht mitgeteilt wird und daß an die Stelle des Rechtsweges die Nachprüfung durch von der Volksvertretung bestellte Organe und Hilfsorgane tritt.

Das Fernmeldegeheimnis verbietet Unbefugten unter anderem das Überwachen von Fernmeldeinformationen. Die Legaldefinition des Fernmeldegeheimnisses steht in § 88 des *Telekommunikationsgesetzes (TKG)* sowie in § 206 des *Strafgesetzbuchs (StGB)*. § 88 des TKG ist im Folgenden aufgeführt:

(1) Dem Fernmeldegeheimnis unterliegen der Inhalt der Telekommunikation und ihre näheren Umstände, insbesondere die Tatsache, ob jemand an einem Telekommunikationsvorgang beteiligt ist oder war. Das Fernmeldegeheimnis erstreckt sich auch auf die näheren Umstände erfolgloser Verbindungsversuche.

(2) Zur Wahrung des Fernmeldegeheimnisses ist jeder Diensteanbieter verpflichtet. Die Pflicht zur Geheimhaltung besteht auch nach dem Ende der Tätigkeit fort, durch die sie begründet worden ist.

(3) Den nach Absatz 2 Verpflichteten ist es untersagt, sich oder anderen über das für die geschäftsmäßige Erbringung der Telekommunikationsdienste einschließlich des Schutzes ihrer technischen Systeme erforderliche Maß hinaus Kenntnis vom Inhalt oder den näheren Umständen der Telekommunikation zu verschaffen. Sie dürfen Kenntnisse über Tatsachen, die dem Fernmeldegeheimnis unterliegen, nur für den in Satz 1 genannten Zweck verwenden. Eine Verwendung dieser Kenntnisse für andere Zwecke, insbesondere die Weitergabe an andere, ist nur zulässig, soweit dieses Gesetz oder eine andere gesetzliche Vorschrift dies vorsieht und sich dabei ausdrücklich auf Telekommunikationsvorgänge bezieht. Die Anzeigepflicht nach § 138 des Strafgesetzbuches hat Vorrang.

Diesem Paragraphen entnehmen wir, dass die Administration eines Netzwerks durchaus berechtigt ist, insbesondere Metainformationen einer Verbindung aufzuzeichnen, da es sich in unserem Falle um den Schutz technischer Systeme handelt. Dabei sind allerdings die Bestimmungen des Datenschutzes im *Bundesdatenschutz- (BDSG)*, *Landesdatenschutz- (LDSG)* und Telekommunikationsgesetz zu beachten. Aus diesen geht hervor, dass besonders ein Speichern von personenbezogenen Daten ein individuelles Einverständnis der betroffenen Personen erfordert. Generell kann man aber von Personen ein Zustimmung verlangen, wenn diese das betroffene Netzwerk verwenden möchte.

Da wir uns nun über die technische und rechtliche Situation des Überwachens im Klaren sind, können wir entsprechende Daten aufzeichnen und auswerten.

5 Analyse der aufgezeichneten Daten

5.1 Vorbereitung der Analyse

Eine ausführliche Analyse der durch die Überwachung erhaltenen Daten ist prinzipiell nur im Verdachtsfall illegitimer Nutzung oder bei Stichproben nötig. Zu Beginn wird es vorteilhaft sein, die gesammelten Daten nach Datum und Uhrzeit des Verdachtfalls beziehungsweise der Stichprobe zu filtern. Auch sollten bekannte harmlose Verbindungen, wenn diese nicht schon bei der Überwachung ausgeschlossen worden sind, aus den angehäuften Daten entfernt werden. Diese Maßnahmen verhindern unnötigen Aufwand bei der weiteren Analyse, an der, wie wir später sehen werden, auch der Mensch direkt beteiligt ist.

Im Folgenden betrachten wir Möglichkeiten, zusätzliche Informationen zu den bereits aufgezeichneten Daten zu erhalten. Dabei schauen wir uns vor allem vom externen Netzwerk ausgehende illegitime Nutzungen an, da sich interne illegitime Verwendungen relativ einfach analysieren lassen.

5.2 Proxys, offene Relays und Zombies

Mit Hilfe der IP-Adresse einer Verbindung, die nicht zum internen Netzwerk gehört, versuchen wir zunächst grob festzustellen, wer illegitime Absichten gegen uns hegt. Hierzu überprüfen wir, ob die IP-Adresse zu einem bekannten Proxy, Mix oder *offenem Relay* gehört (Abbildung 5). Beispielsweise können wir Web-Dienste benutzen oder ausgewählte Ports überprüfen, um dies festzustellen. Selbst wenn die IP-Adresse zu keinem bekannten Proxy oder offenem Relay gehört, kann man nicht unbedingt davon ausgehen, dass die IP-Adresse die des Angreifers ist. So kann die IP-Adresse, die zu einem Computer führt, von einem *Internet Service Provider (ISP)* dynamisch vergeben worden sein oder aber der Rechner wurde vom eigentlichen Angreifer manipuliert und für illegitime Nutzung ohne Wissen des Benutzers missbraucht. Ein solcher Computer wird dann als *Zombie* bezeichnet. In den hier geschilderten Fällen ist der Angreifer leider nicht ohne Einbezug der Justiz weiter lokalisierbar. Wir wissen aber im Falle von Proxys oder offenen Relays, dass ein Angriff nicht unwahrscheinlich ist. Sollte die IP-Adresse nicht auf oben genanntes hinweisen, so lohnen sich in jedem Falle weitere Untersuchungen.

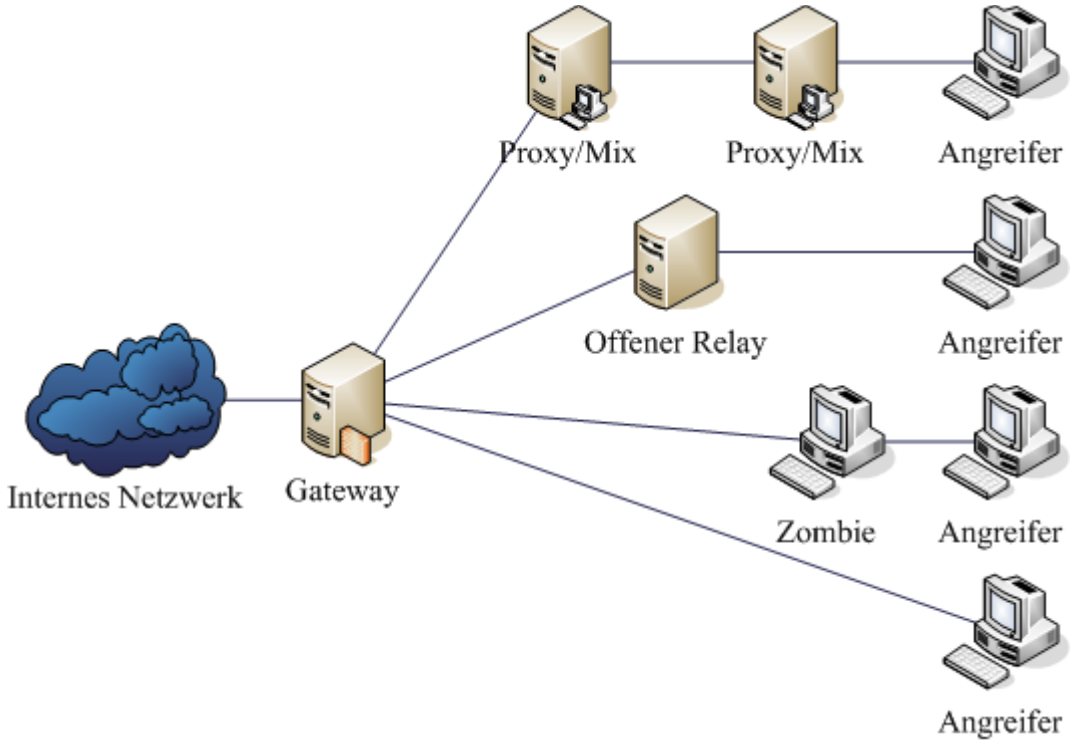

Abbildung 5: Mögliche Angreiferszenarien

5.3 Informationen über IP-Adressen

Eine besonders einfache Methode, Informationen über IP-Adressen zu erlangen, besteht darin, dass *WhoIs-Protokoll* zu verwenden. Hierüber kann man Daten, wie Betreiber und Ort, zu einem zur IP-Adresse zugehörigen Rechner erfahren. Auch lässt sich feststellen, ob eine IP-Adresse zu einem ISP gehört und somit dynamisch sein könnte. Der Ort des Computers liefert für uns einen wichtigen Hinweis. So ist eine Verbindung in ein Land, dessen Justiz nur eingeschränkt gegen die für uns illegitime Nutzung vorgeht, verdächtiger als eine in ein Land mit restriktiver Justiz.

Im Nachstehenden zeigen wir eine simple Implementierung eines *WhoIs-Clients* unter Microsoft Windows in C# .NET 2.0, mit dessen Hilfe wir die oben genannten Informationen von einem *WhoIs-Server* erhalten können:

```
private void WhoIs()
{
      // ein WhoIs-Server
      const string WHOIS_SERVER = "whois.ripe.net";
      // Name oder IP-Adresse, deren Informationen abgefragt werden sollen
      const string WHOIS_NAME = "www.uni-magdeburg.de";

      // ein TCP-Client
      TcpClient tcpC = new TcpClient();
      // mit WhoIs-Server am Port 43 verbinden
      tcpC.Connect(WHOIS_SERVER, 43);
      // Datenstrom für Lesen und Schreiben erhalten
      NetworkStream ns = tcpC.GetStream();

      // Anfrage an WhoIs-Server erstellen
      byte[] request = Encoding.ASCII.GetBytes(
                  Dns.GetHostAddresses(WHOIS_NAME)[0].ToString() + "\r\n");
      // Anfrage senden
      ns.Write(request, 0, request.Length);
```

```
        // Variablen für Antwort-Behandlung
        byte[] readBuffer = new byte[1024];
        StringBuilder response = new StringBuilder();
        int numberOfBytesRead = 0;

        // Antwort vom WhoIs-Server erhalten
        do
        {
            // Teilantwort empfangen
            numberOfBytesRead = ns.Read(readBuffer, 0, readBuffer.Length);
            // Teilantworten zusammensetzen
            response.Append(
                      Encoding.ASCII.GetString(readBuffer, 0, numberOfBytesRead));
        }
        while (numberOfBytesRead != 0);

        // Antwort des WhoIs-Servers ausgeben
        Console.WriteLine(response.ToString());

        // Datenstrom beenden
        ns.Close();
        // Verbindung trennen
        tcpC.Close();
}
```

Das Programm verbindet sich über Port 43 mit dem durch *WHOIS_SERVER* angegebenen WhoIs-Server und übergibt ihm mit *WHOIS_NAME* den Hostnamen beziehungsweise die IP-Adresse des zu überprüfenden Rechners. Der WhoIs-Server sendet dann die gewünschten Informationen, welche vom Programm empfangen und ausgegeben werden.

5.4 Abfragen von Diensten

Das Abfragen von Diensten über die IP-Adresse auf einem Rechner kann durchaus Aufschluss über seine Verwendung geben. So kann man mit Hilfe eines so genannten *Portscanners* offene TCP- oder UDP-Ports, die Hinweise auf betriebene Dienste liefern, erkennen. Generell gilt aber, dass ein beliebiger Dienst auch einen beliebigen Port verwenden kann. Somit muss beispielsweise nicht unbedingt nur ein *HTTP-Server* auf dem TCP-Port 80 erreichbar sein, sondern es können auch Schadprogramme über diesen kommunizieren. Welche Ports im Allgemeinen für bestimmte Dienste verwendet werden dürfen, legt die *Internet Assigned Numbers Authority (IANA, http://www.iana.org)* fest. So liegen im Bereich zwischen 0 und 1023 die Ports für gemeinnützige Dienste, wie etwa *FTP* am TCP-Port 21. Dagegen werden die Ports von 1024 bis 49151 von registrierten Firmen benutzt und die übrigen Ports (49152 - 65535) für private Zwecke. Eine Verwendung der nicht registrierten Ports sollte generell als Anhaltspunkt für illegitime Nutzung betrachtet werden.

Im Folgenden werden wir nun zeigen, wie man relativ problemlos einen Portscanner unter Microsoft Windows mit C# .NET 2.0 implementieren kann. Dabei betrachten wir zuerst die Implementierung eines TCP-Port-Scanners, der sukzessive alle TCP-Ports eines vorgegebenen Bereichs testet:

```
private void TCPPortScan()
{
        // die Ports werden von START_PORT bis END_PORT überprüft
        const int START_PORT = 21;
        const int END_PORT = 80;
        // ein beliebiger Host (z.B. localhost mit 127.0.0.1)
        const string HOST_NAME = "localhost";
```

```
        // IP-Adresse und Port von HOST_NAME (mit DNS-Auflösung)
        IPEndPoint ipepHost = new IPEndPoint(
            Dns.GetHostAddresses(HOST_NAME)[0], 0);
        // ein TCP-Client
        TcpClient tcpC = null;

        // Ports von START_PORT bis END_PORT durchlaufen
        for (int i = START_PORT; i <= END_PORT; i++)
        {
            // aktuellen Port setzen
            ipepHost.Port = i;

            try
            {
                        tcpC = new TcpClient();

                        // versuche Verbindung mit Host am Port i aufzunehmen
                        tcpC.Connect(ipepHost);
                        // Verbindung trennen
                        tcpC.Close();

                        // offenen Port i ausgeben
                        Console.WriteLine(i);
            }
            catch // Verbindungsfehler
            {
                        // Verbindung trennen
                        tcpC.Close();
            }
        }
}
```

Das Programm durchläuft alle zwischen *START_PORT* und *END_PORT* angegebenen Ports und testst dabei mit *tcpC.Connect(...)*, ob eine Verbinding am aktuellen Port mit *HOST_NAME* hergestellt werden kann. Ist dies nicht der Fall, so wird eine Ausnahme geworfen und die Verbindung geschlossen. Sollte allerdings keine Ausnahme auftreten, so wird der aktuelle Port vom Programm ausgegeben. Als nächstes betrachten wir eine mögliche Realisierung eines UDP-Port-Scanners:

```
// wahr, wenn eine Antwort erhalten wurde
private bool messageReceived = false;

// asynchroner Callback für UDPClient.BeginReceive(...)
private void ReceiveCallback(IAsyncResult ar)
{
        // Antwort erhalten
        messageReceived = true;
}

private void UDPPortScan()
{
        // die Ports werden von START_PORT bis END_PORT überprüft
        const int START_PORT = 53;
        const int END_PORT = 111;
        // ein beliebiger Host (z.B. localhost mit 127.0.0.1)
        const string HOST_NAME = "localhost";

        // IP-Adresse und Port von HOST_NAME (mit DNS-Auflösung)
        IPEndPoint ipepHost = new IPEndPoint(
            Dns.GetHostAddresses(HOST_NAME)[0], 0);
        // IP-Adresse und Port eines Host vom Empfang von Daten
```

```
        IPEndPoint ipepReceive = new IPEndPoint(IPAddress.Any, 0);
        // ein UDP-Client
        UdpClient udpC = new UdpClient();
        // timeout für Antwort
        int timeout = 0;

        // Ports von START_PORT bis END_PORT durchlaufen
        for (int i = START_PORT; i <= END_PORT; i++)
        {
            // aktuellen Port setzen
            ipepHost.Port = i;

            try
            {
                        // leeres Paket senden
                        udpC.Send(new byte[0], 0, ipepHost);

                        messageReceived = false;
                        // asynchron auf Antwort warten
                        IAsyncResult ar = udpC.BeginReceive(
                                  new AsyncCallback(ReceiveCallback), null);

                        // timeout zurücksetzen
                        timeout = 0;
                        // solange keine Nachricht empfangen wurde und
                        // eine Sekunde noch nicht vergangen ist
                        while (!messageReceived && timeout != 10)
                        {
                                  // 100 ms warten
                                  System.Threading.Thread.Sleep(100);
                                  timeout++;
                        }

                        // wenn Nachricht erhalten wurde,
                        // dann Schleife weiter durchlaufen
                        // Port i ist dann geschlossen
                        if (messageReceived)
                                  continue;

                        // offenen Port i ausgeben
                        Console.WriteLine(i);
            }
            catch // Fehler ignorieren
            {
            }
        }

        // Verbindung trennen
        udpC.Close();
}
```

Dieses Programm funktioniert im Prinzip so wie der TCP-Port-Scanner. Da UDP allerdings ein *verbindungsloses* Protokoll ist, verläuft die Überprüfung auf offene Ports anders. So sendet das Programm ein leeres Paket an den zu überprüfenden Port. Sollte das Programm nun keine Antwort erhalten, so ist der Port offen. Im anderen Falle ist der Port dann natürlich geschlossen.

Die gegebenen Programme sollen nur einen technischen Eindruck der Analysemöglichkeiten zeigen. Auf dem Markt existiert bereits eine große Anzahl von Werkzeugen dieser Art. GFI *LANguard Network Security Scanner* für Windows ist beispielsweise eine Programm, welches neben selektiven Portscans auch Informationen über das Betriebssystem (*OS-Fingerprinting*) erlangt.

6 Konsequenzen im Falle illegitimer Nutzung

Sollte eine illegitime Handlung vom internen Netzwerk aus erfolgt sein, so kann der entsprechende Benutzer zur Verantwortung gezogen werden und nötige Maßnahmen gegen ihn eingeleitet werden. Geschah die illegitime Nutzung vom externen Netzwerk aus, wird es hingegen schwierig sein eine verantwortliche Person zu bestimmen und uns bleibt somit nur die Möglichkeit zusätzliche präventive Maßnahmen zu ergreifen. So können wir, wie in Abbildung 6 gezeigt, beispielsweise sämtliche IP-Adressen zu Proxys und offenen Relays oder verdächtigen Ports einfach sperren (*blacklisting*).

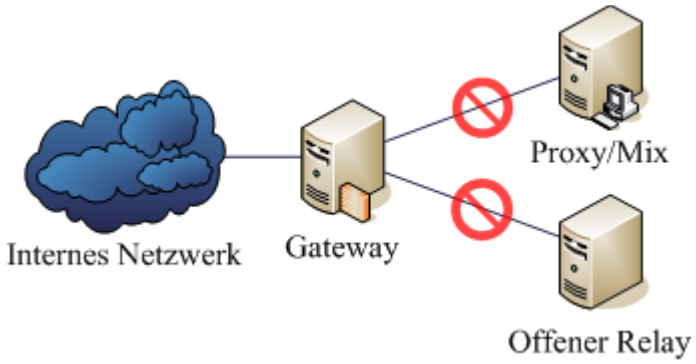

Abbildung 6: Blacklisting

Verweise

http://anon.inf.tu-dresden.de/

http://www.ietf.org/rfc/rfc3912.txt

http://de.wikipedia.org/wiki/IP-Header

http://de.wikipedia.org/wiki/Transmission_Control_Protocol

http://de.wikipedia.org/wiki/UDP

http://de.wikipedia.org/wiki/Anonymizer

http://de.wikipedia.org/wiki/IPv4

http://de.wikipedia.org/wiki/WhoIs

http://de.wikipedia.org/wiki/Portscanner

http://de.wikipedia.org/wiki/OS-Fingerprinting

http://de.wikipedia.org/wiki/Port_%28Protokoll%29

http://de.wikipedia.org/wiki/Sniffer

http://de.wikipedia.org/wiki/Intrusion_Detection_System

http://de.wikipedia.org/wiki/Steganographie

http://de.wikipedia.org/wiki/Tunneling

http://de.wikipedia.org/wiki/Datenschutzrecht

http://de.wikipedia.org/wiki/Fernmeldegeheimnis

http://de.wikipedia.org/wiki/Telekommunikationsgesetz

http://de.wikipedia.org/wiki/Bundesdatenschutzgesetz

Grundgesetz für die Bundesrepublik Deutschland

Telekommunikationsgesetz (TKG)

MSDN Library for Visual Studio 2005