KAPITEL 2

Windows Reverse Engineering

Das Software Reverse Engineering, auch *Reverse Code Engineering* (RCE) genannt, ist die Kunst des Sezierens unbekannter Binäranwendungen, deren Quellcode nicht offen liegt (»closed source«). Im Gegensatz zur Open Source-Software, die theoretisch einfacher auf Sicherheitslücken untersucht werden kann, präsentiert Closed Source-Software dem Benutzer eine »Blackbox«. Historisch gesehen wurde RCE auf Windows-Plattformen durchgeführt, aber es gibt nun auch einen wachsenden Bedarf an professionellem Linux-Reverse-Engineering, wie wir in Kapitel 3 noch erläutern werden.

Das RCE erlaubt es Ihnen, in diese Blackbox hineinzusehen. Indem Sie eine Binäranwendung disassemblieren, können Sie die Ausführung des Programms auf unterster Ebene beobachten. Sobald eine Anwendung auf Maschinensprache heruntergebrochen werden konnte, kann ein erfahrener Praktiker die Operation jeder Binäranwendung verfolgen, ganz egal, wie gut der Software-Entwickler sie zu schützen versucht hat.

Warum sollte man als Sicherheitsexperte RCE lernen wollen? Der Hauptgrund ist das Reversing von Malware wie Viren und Trojanern. Die Antivirus-Industrie ist von der Fähigkeit abhängig, Binaries zu analysieren, um diese Schädlinge diagnostizieren, desinfizieren und verhindern zu können. Darüber hinaus führt die Verbreitung von ethisch fragwürdiger kommerzieller Spyware und von »nach Hause telefonierenden« Kopierschutzmechanismen zu ernsthaften Verletzungen der Privatsphäre.



In diesem Kapitel arbeiten wir mit Windows-Betriebssystemen für den Desktop. Weil Windows eine geschlossene und häufig feindselige Plattform ist, hat Darwinscher Druck dazu geführt, dass das Windows-RCE die Spitze der Entwicklung darstellt. In den nachfolgenden Kapiteln stellen wir RCE für andere Plattformen vor, darunter Linux und Windows CE, wo sich RCE noch in den Kinderschuhen befindet.

Die Legalität des RCE ist in vielen Bereichen immer noch fraglich. Ein Großteil der kommerziellen Software wird mit einer Endbenutzer-Lizenzvereinbarung (end-user license agreement, kurz EULA) ausgeliefert, durch die man sich einfach nur »durchklicken« muss. Nach Ansicht der Software-Hersteller akzeptieren Sie deren Lizenzbedingungen,

sobald Sie bei der Software-Installation »Ich stimme zu« anklicken. Die meisten EULAs enthalten eine Klausel, die dem Endbenutzer (mit Hinweis auf den Schutz intellektuellen Eigentums des Herstellers) ein Reverse Engineering der Anwendung untersagen. Tatsächlich sieht der amerikanische Digital Millennium Copyright Act (DMCA) für einige Arten des Reverse Engineering harte Strafen vor.[1]

Zum Beispiel waren wir als Konferenzredner auf der Defcon 9 zur Computersicherheit schockiert und verzweifelt, hören zu müssen, dass einer unser Kollegen verhaftet wurde, nur weil er über seine akademischen Forschungen sprach. Nach seiner Rede über die Sicherheit von E-Books wurde Dmitry Sklyarov, ein 27 Jahre alter russischer Doktorand, auf dem Gelände des Alexis Park Hotels verhaftet. Die vom FBI durchgeführte Verhaftung wurde durch eine Beschwerde von Adobe Systems veranlasst, dem Hersteller der fraglichen E-Book-Software.

In einem Schritt, der einen neuen Präzedenzfall zu schaffen schien, behauptete der ausführende FBI-Agent bei der Beantragung des Haftbefehls, dass die Defcon als »Hacker«-Konferenz angekündigt worden sei und dass die Sprecher daher Kriminelle sein müssten. Allerdings vergaß dieser FBI-Agent zu erwähnen, dass andere hochrangige Gesetzesvertreter, Mitglieder des Militärs und sogar einige FBI-Kollegen auf der gleichen »Hacker«-Konferenz (und deren Vorgänger Black Hat) als Sprecher angekündigt waren. Tatsächlich sprach Richard Clarke, Sonderberater von Präsident Bush für die Sicherheit im Cyberspace, im folgenden Jahr auf der Defcon.

Sklyarov half seinem russischen Arbeitgeber Elcomsoft bei der Entwicklung des Advanced eBook Processor (AEBPR). Laut Elcomsoft ermöglicht es diese Software den Besitzern von E-Books, das sichere E-Book-Format von Adobe in das weit verbreitete Portable Document Format (PDF) umzuwandeln. Da die Software nur mit legal erworbenen E-Books funktioniert, unterstützt sie nicht direkt die Verletzung von Urheberrechten. Die Software ist nützlich, um gültige Datensicherungen wertvoller Daten anzulegen.

Sklyarov wurde angeklagt, ein Produkt vertrieben zu haben, das entwickelt wurde, um ein Kopierschutzverfahren zu umgehen, was laut DMCA (auf das wir später in diesem Kapitel eingehen) eine Straftat darstellt. Ein breiter Aufschrei ging durch die akademische Welt und die Anhänger der individuellen Handlungsfreiheit, und die Proteste machten sich sogar weltweit vor den Büros von Adobe Luft. Adobe, das seinen gravierenden Fehler zu ahnen begann, ruderte sofort zurück – aber es war zu spät. Der Schaden war schon passiert.

Sklyarov wurde schließlich gegen eine Kaution von $50.000 freigelassen und durfte Kalifornien nicht verlassen. Im Dezember 2001 durfte er mit seiner Familie nach Russland

1 Auch in Europa kann jeder nicht gestattete Eingriff in eine Software strafbar sein. Abgesehen von zivilrechtlichen Ansprüchen der Hersteller ist Reverse Software Engineering in Deutschland (wie in der ganzen EU) prinzipiell verboten. Siehe hierzu unter anderem das »Gesetz über Urheberrecht und verwandte Schutzrechte« (UrhG, *http://bundesrecht.juris.de/bundesrecht/urhg/*) mit Stand vom September 2003. Relevant sind darin vor allem: Teil 1, Urheberrecht, Abschnitt 8, Besondere Bestimmungen für Computerprogramme. In § 69e, Dekompilierung, werden die Bedingungen genannt, unter denen Reverse Engineering erlaubt ist (zur »Interoperabilität mit anderen Programmen«). – Anm. d. V.

zurückkehren, allerdings nur unter der Bedingung, auf Abruf in die USA zurückzukehren und gegen seinen Arbeitgeber Elcomsoft auszusagen. Nach einem ärgerlichen Rechtsstreit wurden sowohl Sklyarov als auch Elcomsoft schließlich vollständig rehabilitiert.

Der DMCA gibt einem immer noch einen gewissen Spielraum, da das Gesetz »Sicherheitsexperten« in eingeschränktem Maß erlaubt, Schutzmechanismen zu umgehen, um deren Sicherheit zu prüfen. Allerdings ist die Interpretation dieser Klausel nach wie vor sehr nebulös.

Geschichte des RCE

Das »moderne« RCE begann mit Programmierern, die den Kopierschutz klassischer Computerspiele umgingen, etwa solchen, die in den frühen 1980ern für den Apple II entwickelt wurden. Auch wenn dieser Trend dazu missbraucht wurde, raubkopierte Software zu vertreiben, blieb doch ein Kern von Experten übrig, die sich dem Thema RCE allein aus akademischen Gründen widmeten.

Eine der legendären Figuren dieser spannenden Zeit war Old Red Cracker (+ORC). +ORC war aber nicht nur ein genialer Software-Reverser, sondern auch ein produktiver Autor und Lehrer. Seine klassischen Texte werden heute noch als grundlegende Lektüre für alle RCE-Interessierten betrachtet.

Um die RCE-Entwicklung voranzubringen, gründete +ORC die High Cracking University (kurz +HCU). Das »+«-Zeichen oder »Handle« neben dem Namen war ein Erkennungszeichen der Mitglieder der +HCU. Zu den +HCU-»Studenten« zählte ein Großteil der weltweit besten Windows-Reverser. Jedes Jahr veröffentlichte die +HCU eine neue Reverse-Engineering-Aufgabe (eine so genannte »Challenge«), und die Autoren einer Handvoll der besten Antworten wurden als Studenten für das neue Schuljahr eingeladen.

Einer der Professoren, bekannt als +Fravia, pflegte eine bunt gemischte Website namens »+Fravia's Pages of Reverse Engineering«. In diesem Forum stellte +Fravia Programmierern nicht nur Challenges zur Verfügung, sondern forderte die Gesellschaft selbst auf, die Gehirnwäsche eines korrupten und wild wuchernden Materialismus einem »Reverse Engineering« zu unterziehen. Auf dem Höhepunkt wurden auf +Fravias Site abermillionen Hits pro Jahr verzeichnet, und sein Einfluss war überall spürbar.

Heutzutage hat ein Großteil der alten +HCU-Recken Windows den Rücken gekehrt, um sich der weniger okkulten Linux-Plattform zu widmen. Nur einige wenige wie +Tsehp beschäftigen sich noch mit dem Reverse Engineering von Windows-Software. Eine neue Generation von Reversern hat die alten Texte wiederentdeckt und damit begonnen, die alte Wissenschaft wieder voranzubringen. +Fravia selbst kann man noch immer in seiner endlosen Bibliothek unter *http://www.searchlores.org* herumwandern sehen.

Reversing-Tools

Als Software Reverse Engineer sind Sie nur so gut wie die von Ihnen verwendeten Tools. Bevor wir uns später praktischen Beispielen zuwenden, wollen wir uns einige der klassischen Windows RCE-Tools ansehen. Einige kann man an einem Tag erlernen, während man für andere Jahre braucht, um sie zu beherrschen.

Hex-Editoren

Um Binaries im Hex(adezimal)-Code editieren zu können (das so genannte *Opcode-Patching*), benötigen Sie einen guten Hex-Editor. Einer der besten ist UltraEdit von Ian Meade (*http://www.ultraedit.com/*), zu sehen in Abbildung 2-1.

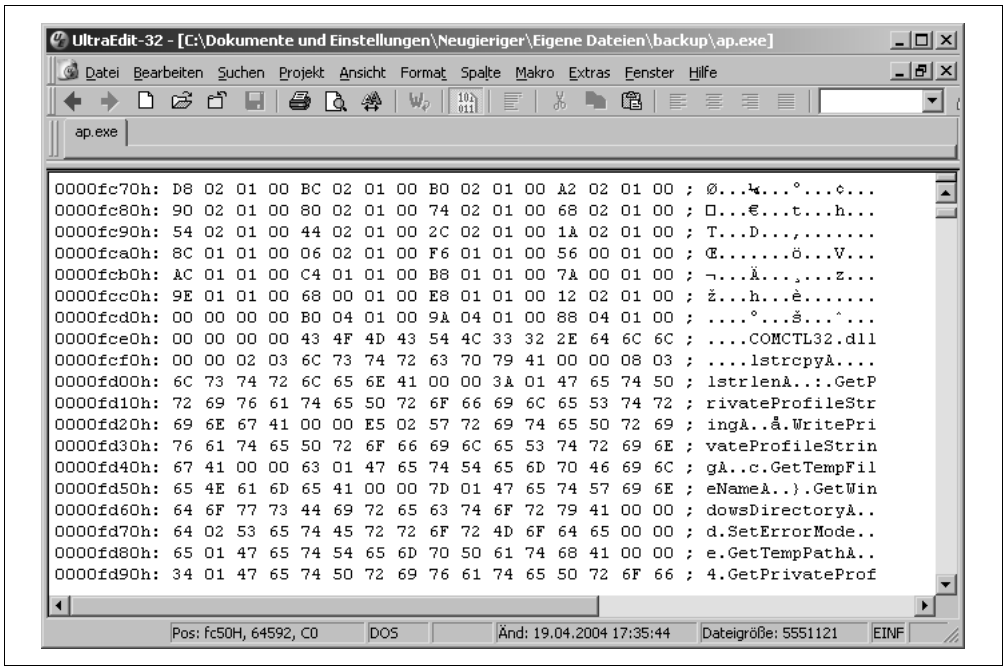


Abbildung 2-1: Für das Opcode Patching empfehlen wir UltraEdit, einen fortschrittlichen Windows-Hex-Editor

Disassembler

Ein Disassembler versucht, ein Executable in für uns Menschen verständlichen Assembler-Code umzuwandeln. Die Disassembler-Software liest den Bytestream ein und interpretiert ihn in Gruppen von Befehlen. Diese Befehle werden dann in Assembler-Anweisungen umgewandelt. Der Disassembler versucht sein Bestes, um vernünftigen Assembler-Code zu erzeugen, häufig mit wechselndem Erfolg. Dennoch ist er das elementarste Werkzeug des Software-Crackers.

Ein weit verbreiteter Disassembler – und eines der bevorzugten Tools vieler erfahrener Reverse – ist IDA Pro. IDA (*http://www.datarescue.com*) ist ein interaktiver Disassembler, der eine Vielzahl von Prozessoren und Betriebssystemen unterstützt. Er hat eine Vielzahl von Preisen gewonnen und wurde unter anderem 1997 zum offiziellen Disassembler der +HCU gewählt.

IDA betrachtet eine ausführbare Datei als strukturiertes Objekt, das aus einer den Quellcode repräsentierenden Datenbank erzeugt wurde. Er versucht also mit anderen Worten, den gesamten Quellcode vollständig wiederherzustellen (im Gegensatz zu W32DASM, das nur den Code ausgibt, den es für wichtig hält).

Eines der leistungsfähigsten Features von IDA ist die Verwendung so genannter FLIRT-Signaturen. FLIRT steht für »Fast Library Identification and Recognition Technology« und stellt einen von IDA verwendeten, proprietären Algorithmus dar, der versucht, Compiler-spezifische Bibliotheksfunktionen zu erkennen.

IDA zu meistern verlangt viel Zeit und Mühe. Das Unternehmen gibt in seinem Benutzerhandbuch unumwunden zu, dass IDA schwierig zu verstehen ist. Sobald Sie IDA aber einmal beherrschen, werden Sie es wahrscheinlich der Kombination aus W32DASM und SoftICE (die als Nächstes besprochen wird) vorziehen. Dieser Abschnitt stellt einige grundlegende Schritte zur Konfiguration und Anpassung von IDA vor.

Eine Konfigurationsdatei kontrolliert die IDA-Präferenzen. Suchen Sie in Ihrem *Programmdateien*-Verzeichnis nach dem *IDA*-Ordner und öffnen Sie *Ida.cfg* (die Konfigurationsdatei) mit einem Texteditor. Die Konfigurationsdatei wird zweimal eingelesen. Das erste Mal beim Laden von IDA und das zweite Mal, wenn IDA den Prozessortyp bestimmt. Alle prozessorspezifischen Anpassungen stehen im zweiten Teil der Konfigurationsdatei.

IDA erlaubt beim Start die Auswahl eines Standard-Prozessors. Wie Sie in Beispiel 2-1 sehen können, wird eine Vielzahl unterschiedlicher Prozessortypen unterstützt. Hier können Sie sich die von IDA unterstützten Prozessoren ansehen. Wenn Sie zum Beispiel hauptsächlich mit PocketPC-Anwendungen (Windows CE) arbeiten, werden Sie sehr wahrscheinlich den ARM-Prozessor verwenden. Anderenfalls ist die Standardeinstellung `"metapc"` (x86).

Beispiel 2-1: Prozessorspezifische Parameter in IDA Pro

```
/* Extension    Processor */
  "com" :       "8086"                  // IDA probiert die angegebenen
  "exe" :       "metapc"                // Erweiterungen aus, wenn keine
  "dll" :       "metapc"                // Erweiterung angegeben wurde.
  "drv" :       "metapc"
  "sys" :       "metapc"
  "bin" :       "metapc"
  "ovl" :       "metapc"
  "ovr" :       "metapc"
  "ov?" :       "metapc"
  "nlm" :       "metapc"
```

Beispiel 2-1: Prozessorspezifische Parameter in IDA Pro (Fortsetzung)

```
  "lan" :       "metapc"
  "dsk" :       "metapc"
  "obj" :       "metapc"
  "prc" :       "68000"                 // PalmPilot-Programme
  "axf" :       "arm710a"
  "h68" :       "68000"                 // MC68000 für *.H68-Dateien
  "i51" :       "8051"                  // i8051   für *.I51-Dateien
  "sav" :       "pdp11"                 // PDP-11  für *.SAV-Dateien
  "rom" :       "z80"                   // Z80     für *.ROM-Dateien
  "cla*":       "java"
  "s19":        "6811"
  "o":          "metapc"
```

IDA erlaubt die Feineinstellung diverser Disassembler-Optionen. Zum Beispiel können Sie festlegen, ob 90h NOPs automatisch analysiert werden. Die entsprechende Konfiguration ist in Beispiel 2-2 zu sehen.

Beispiel 2-2: IDA-Optionen für die Disassemblierung

```
#ifdef __PC__                           // INTEL 80x86-PROZESSOREN
USE_FPP                 = YES
                                        // Befehle des Fließkomma-
                                        // Prozessors aktivieren.

// IBM PC-spezifische Optionen

PC_ANALYSE_PUSH         = YES           // Wandelt unmittelbaren Operanden
                                        // von "push" in einen Offset um.
                                        //
                                        //      Bei der Sequenz
                                        //
                                        //              push    seg
                                        //              push    num
                                        //
                                        //      versucht IDA, <num> in einen
                                        //      Offset umzuwandeln.
                                        //
PC_ANALYSE_NOP          = NO            // Wandelt db 90h nach
                                        // "jmp" in "nop" um.
                                        // Mittlerweile lässt man die Option
                                        // besser weg, da die letzte
                                        // Analysephase eine intelligentere
                                        // Form der Umwandlung von
                                        // 90h in nops verwendet.
                                        //
                                        //
                                        //      Die Sequenz
                                        //
                                        //      jmp     short label
                                        //      db      90h
                                        //
                                        //      wird also zu
```

Beispiel 2-2: IDA-Optionen für die Disassemblierung

```
//
//      jmp     short label
//      nop
```

Nun können Sie IDA starten. Führen Sie das Programm aus und wählen Sie das gewünschte Binary aus. Abbildung 2-2 zeigt das IDA-Startfenster.

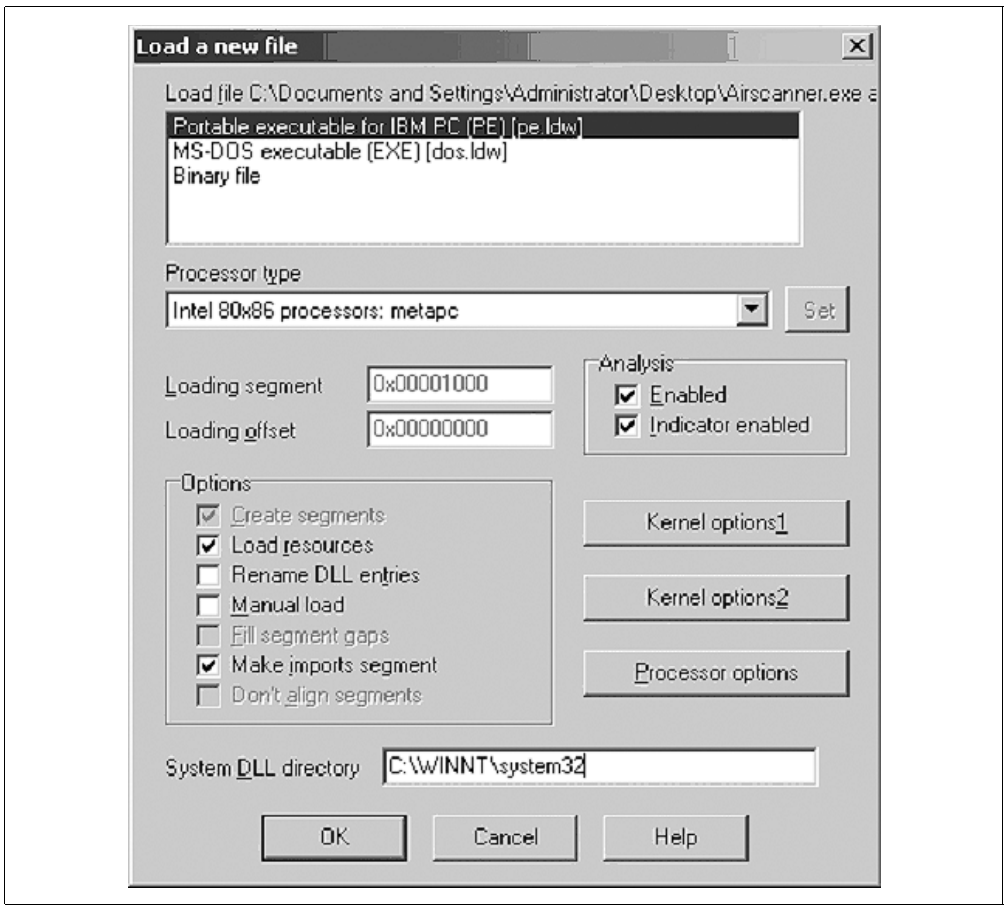


Abbildung 2-2: IDA-Startfenster

Bei den meisten Windows-Dateien werden Sie das Format Portable Executable (PE) verwenden (das wir später noch behandeln werden), weshalb wir auch diese Option wählen. Wählen Sie den Prozessortyp, wenn Sie ihn in Ihrer Konfigurationsdatei nicht bereits als Standard festgelegt haben. Stellen Sie sicher, dass beide ANALYSIS-Optionen aktiv sind. Stellen Sie sicher, dass unter OPTIONS die Optionen »Load resources« und »Make imports segment« aktiviert sind. Stellen Sie außerdem sicher, dass »Rename DLL entries« und »Manual load« nicht aktiviert sind.



Sie müssen sicherstellen, dass bei der Konfiguration von IDA Pro das richtige System-DLL-Verzeichnis angegeben wird.

Wenn Sie fertig sind, klicken Sie auf OK und schauen IDA bei seiner Arbeit zu.

Um sich bei IDA Strings anzusehen, wählen Sie VIEW → OPEN SUBVIEWS → STRINGS (Abbildung 2-3). Sie sehen auch die Optionen für die anderen Unteransichten. Das Tastaturkürzel für Strings ist Shift-F12. Nehmen Sie sich etwas Zeit, um dieses Beispiel-Disassembly zu untersuchen und um sich mit der Navigation in IDA vertraut zu machen.

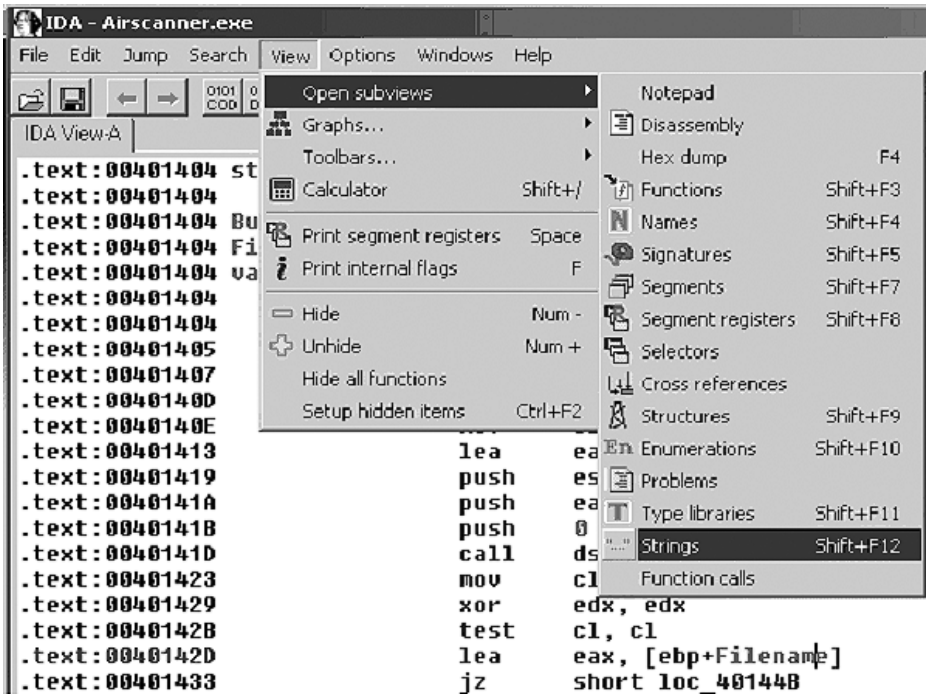


Abbildung 2-3: Strings in IDA betrachten

Debugger

+Fravia nennt SoftICE (*http://www.numega.com*) das »Alpha und das Omega« der Debugger. Die meisten Reverser sind (wenn ihre Haare nicht gerade genauso grau sind wie unsere) wahrscheinlich zu jung, um sich an den Vorläufer von SoftICE, ICE-86, zu erinnern. Hierbei handelte es sich um einen hardwarebasierten In-Circuit-Emulator von Intel, der zum Debugging des in der Entwicklung befindlichen 8086-Prozessors entwickelt wurde. Eine vollständige Beschreibung dieser Hardware finden Sie im klassischen *8086 Family User's Manual*, das 1979 von Intel veröffentlicht wurde.

SoftICE erlaubt die Einzelschrittverarbeitung des Programmcodes sowie das Editieren von Speicher, Registern, Variablen und Flags während der Programmausführung. Die folgenden Funktionstasten erlauben die Einzelschrittverarbeitung und das Bearbeiten des Speichers in SoftICE:

F8
: Einzelschritt.

F10
: Programmschritt.

F11
: Rückkehr aus Subroutine.

F12
: Weiter zum nächsten Rückkehrpunkt.

D
: Speicherinhalt ausgeben.

S
: Speicher nach String absuchen.

WW
: Register überwachen.

Sobald Sie SoftICE installiert haben, bootet Ihr System *WINICE.EXE* zusammen mit Windows. SoftICE ist im Windows-Betriebssystem an Ring 0 installiert, was es so leistungsfähig macht. SoftICE wird über die Datei *WINICE.DAT* konfiguriert. Entfernen Sie die Semikolons in den entsprechenden Zeilen von *WINICE.DAT*, um von Ihnen benötigte Features zu aktivieren. Soll Ihre *WINICE.DAT* zum Beispiel 32-Bit-Aufrufe unterstützen (empfohlen), entfernen Sie das Kommentarzeichen aus folgenden Zeilen:

```
gdi32.dll
kernel32.dll
user32.dll
```

SoftICE ist eine komplexe Anwendung. Tatsächlich wird es mit einem zwei Bände umfassenden Benutzerhandbuch ausgeliefert, das Sie nur mit den Grundlagen seiner Verwendung vertraut machen soll. Das Schwierigste bei der Verwendung von SoftICE ist aber, sich die Befehlskürzel zu merken. Wenn Sie RCE mit SoftICE durchführen, werden Sie eine Referenzliste benötigen, die Ihnen bei der Arbeit zur Verfügung steht. Selbst das offizielle SoftICE-Benutzerhandbuch führt diese kritischen Breakpunkte nicht auf. Aus diesem Grund haben wir eine grundlegende Liste nützlicher SoftICE-Befehle und -Breakpunkte im Anhang aufgeführt. Wir empfehlen Ihnen auch, das SoftICE-Benutzerhandbuch mindestens einmal durchzulesen, bevor Sie die Beispiele am Ende dieses Kapitels durcharbeiten.

System-Monitore

Die Experten von SysInternals (*http://www.sysinternals.com*) haben zwei leistungsfähige Echtzeit-System-Monitore entwickelt: regmon und filemon. Die Programme sind zum persönlichen Gebrauch kostenlos verfügbar (samt Quellcode) und können von deren Website heruntergeladen werden. Mit diesen beiden Programmen können Sie feststellen, welche verdeckten Registry- und Datei-Aufrufe Ihr Ziel-Binary durchführt. Beide Programme sind einfach zu meistern.

Um filemon zu nutzen, installieren Sie es und führen es dann aus. Eine Flut von Daten läuft über das filemon-Fenster, die Sie schnell überwältigen werden. Wir wollen uns hier auf eine zu überwachende Anwendung, zum Beispiel *NOTEPAD.exe* (Abbildung 2-4), konzentrieren.

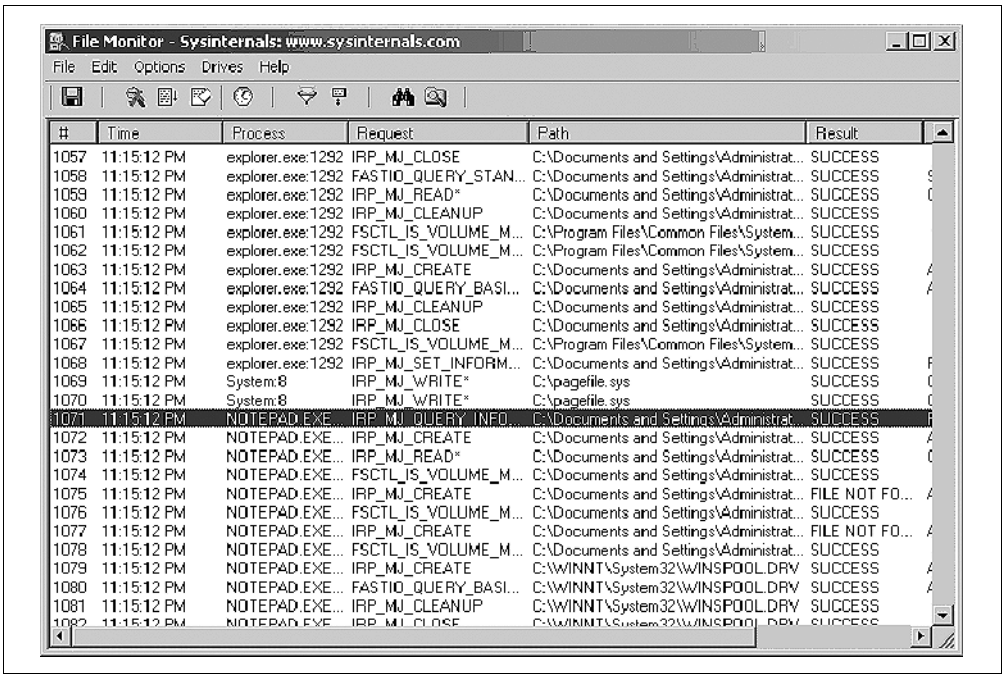


Abbildung 2-4: filemon hält standardmäßig alle Dateizugriffe des Systems fest

Direkt nach dem Start der Zielanwendung drücken Sie Strg-E, um das Sammeln der Daten (Capturing) zu unterbrechen. Bewegen Sie sich nach oben oder unten, bis Sie die gewünschte *.exe*-Datei finden, und drücken Sie dann Strg-L, um sie in das Filter-Fenster zu übernehmen (Abbildung 2-5).

Als Nächstes drücken Sie Strg-X, um den Bildschirm zu löschen, und schalten das Capturing mit Strg-E wieder ein. Wie Sie sehen werden, konzentriert sich das Capturing jetzt auf die Dateizugriffe einer Datei – in diesem Fall *NOTEPAD.exe* (Abbildung 2-6).

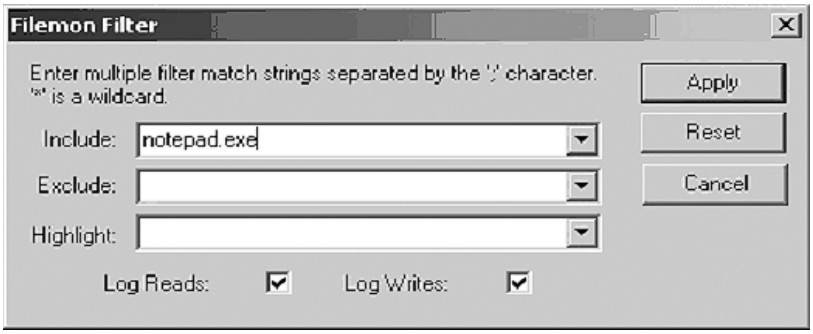


Abbildung 2-5: filemon-Filter verwenden

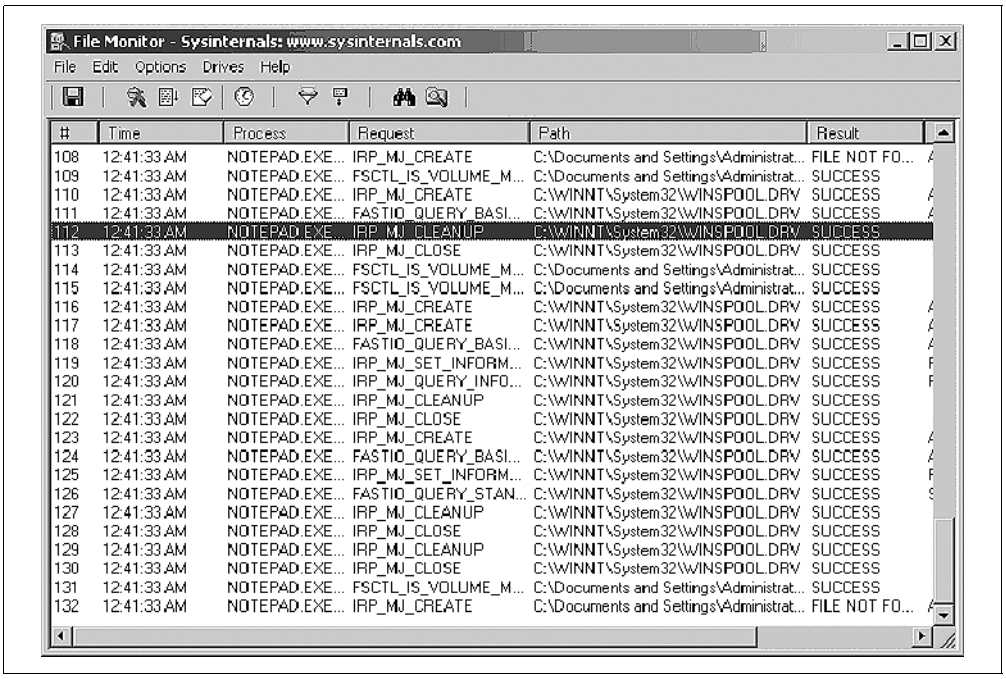


Abbildung 2-6: Gefilterte Ausgabe der NOTEPAD.exe-Dateiaufrufe

Für regmon ist der Prozess nahezu identisch (Abbildung 2-7). Mit regmon können Sie sich zum Beispiel bei einem potenziellen Trojaner ansehen, welche verdeckten Registry-Aufrufe er vornimmt.

Entpacker

Viele kommerzielle Software-Pakete werden mit kommerziellen »Packern« (wie zum Beispiel AsPack von *http://www.aspack.com*) komprimiert, um Platz zu sparen und um

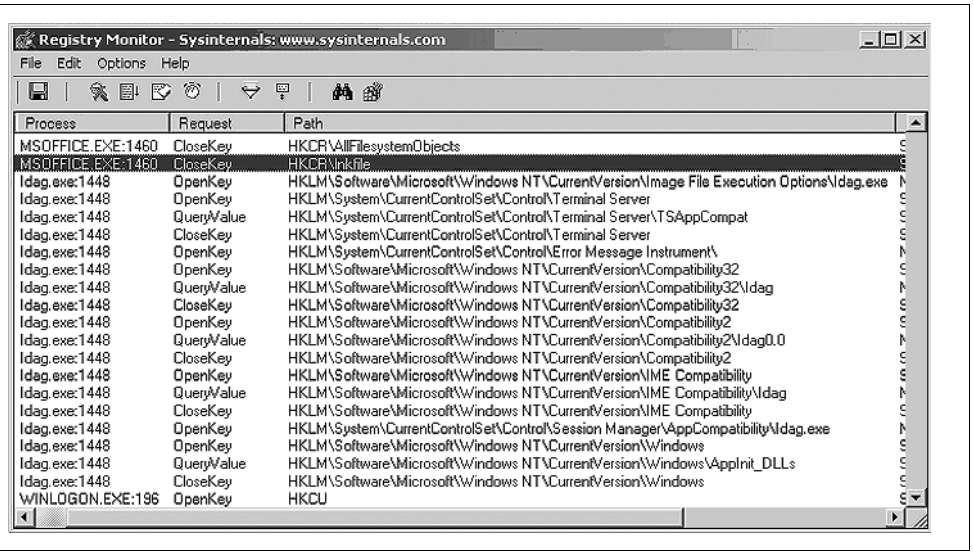


Abbildung 2-7: Verdeckte Registry-Aufrufe mit regmon verfolgen

Disassembler zu frustrieren. Unglücklicherweise sind Sie nicht in der Lage, ein Binary zu disassemblieren, wenn es gepackt ist. Glücklicherweise gibt es Tools zum Entpacken gepackter Binaries. Dieser Abschnitt stellt die Tools und Methoden vor, die zum Entpacken komprimierter Anwendungen verwendet werden. Erst dann ist ein Reverse Engineering möglich.



Die Wissenschaft des Entpackens komprimierter Binaries ist sehr komplex und bildet eine eigene Unterdisziplin des RCE.

Das Dateiformat PE

Das Windows-eigene Dateiformat nennt sich *Portable Executable* (PE). »Portabel« bedeutet, dass das Programm von allen Windows-Plattformen und Prozessoren erkannt wird. Um den Prozess des Entpackens einer komprimierten Anwendung begreifen zu können, müssen Sie zuerst die Struktur des Win32 PE-Dateiformats verstehen (Abbildung 2-8). Dieses Format ist über die Jahre relativ konstant geblieben, selbst bei den neueren 64-Bit-Windows-Plattformen.

Der Assembler oder Compiler des Programmierers erzeugt die PE-Abschnitte automatisch. Die Aufgabe des *DOS MZ-Headers* besteht darin, das Programm unter DOS (Disk Operating System) erkennen zu können. Im Gegensatz dazu ist der *DOS-Stub* einfach ein fest eingebautes Executable, das eine Fehlermeldung (wie »This program cannot be run in MS-DOS mode«) ausgibt, wenn kein Windows auf dem System läuft.

Wir sind hauptsächlich am dritten Abschnitt, dem *PE-Header*, interessiert. Diese Struktur enthält verschiedene vom *PE-Loader* verwendete Felder. Führen Sie das Programm

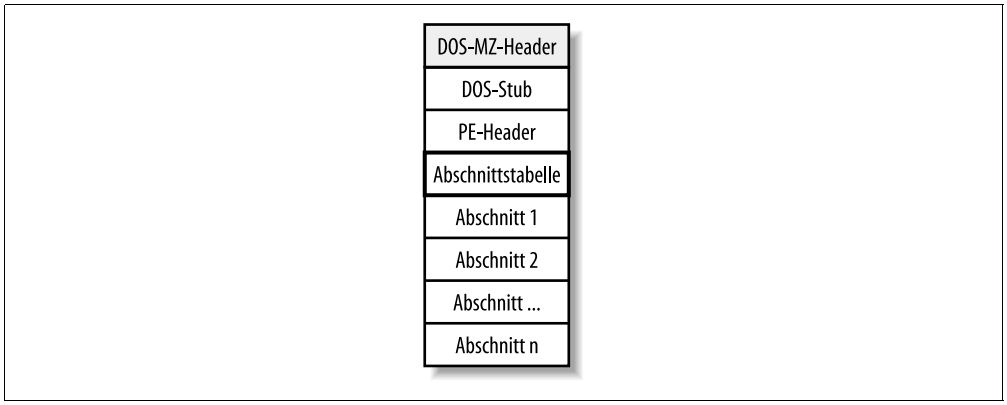


Abbildung 2-8: Vereinfachte Darstellung des PE-Dateiformats

auf einem Betriebssystem aus, das das PE-Dateiformat verarbeiten kann, dann verwendet der PE-Loader den DOS MZ-Header, um den Anfang des PE-Headers zu finden (das heißt, um den DOS-Stub zu überspringen).

Die Daten einer PE-Datei sind in Blöcken gruppiert, die als *Abschnitte* (sections) bezeichnet werden. Diese Abschnitte werden nicht anhand einer logischen Basis gruppiert, sondern basierend auf gemeinsamen Attributen. Ein Abschnitt kann daher sowohl Code als auch Daten enthalten, solange diese die gleichen Attribute aufweisen.

Auf den PE-Header folgt ein als *Abschnittstabelle* (section table) bezeichnetes Array von Strukturen. Jede Struktur enthält abschnittspezifische Daten wie Attribut, Datei-Offset und virtueller Offset.

Während der Programmausführung bildet der PE-Header jeden Abschnitt, basierend auf den in den Abschnitten gespeicherten Informationen, im Speicher ab. Er weist jedem Abschnitt im Speicher Attribute zu, die auf Informationen aus der Abschnittstabelle basieren. Nachdem die PE-Datei im Speicher abgelegt wurde, importiert der PE-Loader Daten aus einem Array, das als die *Importtabelle* bezeichnet wird.

ProcDump

Allein des Lerneffekts wegen werden Sie irgendwann lernen wollen, wie man ein unbekanntes Binary von Hand entpackt. Die RCE-Szene hat allerdings nützliche Tools entwickelt, die viele kommerzielle Packer kennen und Ihnen so viel Zeit sparen. (Stellen Sie sicher, dass Sie die Erlaubnis der jeweiligen Software-Hersteller haben, bevor Sie deren Code einem Reverse Engineering unterziehen.) Zusätzlich gibt es Tools, die Ihnen dabei helfen, selbst *unbekannte* Komprimierungsschemata zu entpacken. ProcDump, entwickelt von G-RoM, Lorian und Stone, ist ein leistungsfähiges Werkzeug, das Sie beim Entpacken unterstützt. Abbildung 2-9 zeigt das Startfenster, das offene Tasks und Module aufführt. Klicken Sie einfach auf UNPACK, um den Unpack-Assistenten zu starten.

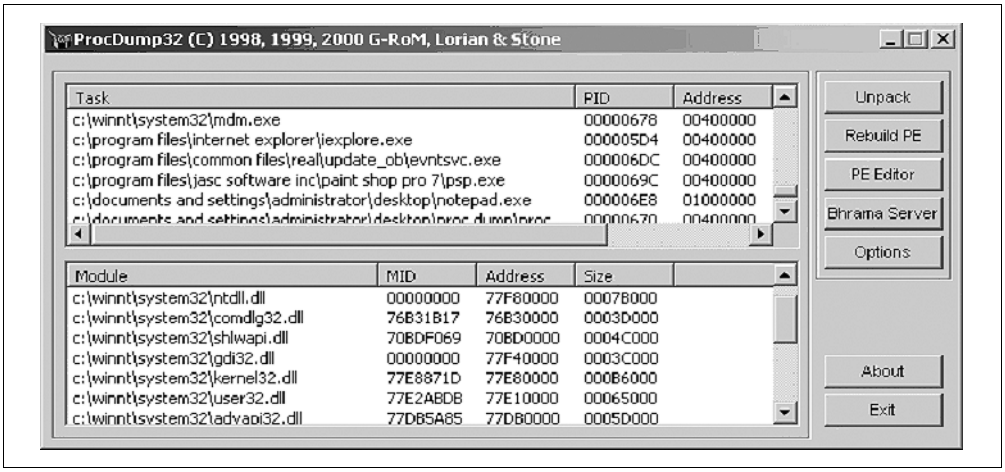


Abbildung 2-9: ProcDump zum Entpacken eines komprimierten Programms nutzen

Nach dem Start von ProcDump erscheint ein zweigeteiltes Fenster. Die obere Hälfte enthält eine Liste aller unter Windows laufenden Prozesse, während die untere Hälfe alle zu einem bestimmten Prozess gehörenden Module zeigt. Auf der rechten Seite des Fensters sehen Sie sechs Buttons:

Unpack
: Entpackt ein Executable oder eine Dump-Datei

Rebuild PE
: Baut die PE-Header eines Executable oder einer Dump-Datei wieder auf

PE Editor
: Ermöglicht das Editieren eines PE-Headers

Bhrama Server
: Startet den Bhrama-Server (der die Entwicklung eigener Plugins für ProcDump erlaubt)

About
: Informationen über die Anwendung

Exit
: Beendet ProcDump

Um eine Anwendung zu entpacken, klicken Sie auf den UNPACK-Button. Dann wählen Sie den Namen des (kommerziellen) Packers, der das Programm schützt. Als Nächstes erscheint ein Dialog zum Dateiöffnen. Wählen Sie das zu entpackende Executable und klicken Sie auf OPEN.

ProcDump lädt das Executable in den Speicher. Sobald das geschehen ist, klicken Sie auf OK, und das Programm wird automatisch entpackt.

Personal Firewalls

Eine Personal Firewall ist eine nützliche Erweiterung des Arsenals eines Reverse Engineer. Personal Firewalls sind auf Endbenutzer-Maschinen laufende Anwendungen, die durch den TCP/IP-Stack laufende Daten filtern. Ist zum Beispiel eine Hintertür (Backdoor) auf Ihrem System installiert, kann eine gute Personal Firewall Sie über die normalerweise verdeckt ablaufende Kommunikation informieren. In gleicher Weise kann eine Personal Firewall kommerzielle Spyware enttarnen, die versucht »nach Hause zu telefonieren«. Beachten Sie, dass Sie aber immer noch ausgetrickst werden können, da einige Produkte mit Port-Redirection/Tunneling arbeiten oder so einfache Methoden verwenden wie etwa das Einbinden des Signals in SMTP-Nachrichten. Ein Beispiel für eine Personal Firewall ist Zone Alarm von *http://www.zonelabs.com*.



Ein Sniffer ist ein weiteres nützliches Tool für den Reverse Engineer. Wir behandeln die Paketanalyse in Kapitel 6.

Installationsmanager

Installationsmanager sind Programme, die unbekannte Binaries überwachen, während diese auf Ihrem System installiert werden. Es gibt eine Vielzahl kommerzieller Installationsmanager wie In Control 5 (Abbildung 2-10).

Eine mögliche Arbeitsweise von Installationsmanagern besteht darin, einen »Schnappschuss« der Laufwerke, Startup-Dateien und Registry-Schlüssel vor und nach der Installation zu vergleichen (Abbildung 2-11).

Wie Sie sehen können, sind Installationsmanager nützlich, um verdeckte Änderungen des Systems während der Installation zu erkennen. Sie sind ganz besonders nützlich, wenn Sie Änderungen durch Spyware und Trojaner verfolgen wollen, um eventuell eigene Schritte zum Desinfizieren unternehmen zu können. Starten Sie einfach den Uninstall-Manager, navigieren Sie zu dem zu installierenden Programm und verwenden dann den Uninstall-Manager, um den Installer zu starten.

Reverse-Engineering-Beispiele

Bevor wir uns der Praxis zuwenden, wollen wir noch einen letzten Hinweis loswerden. Wie das Software-Debugging erfolgt auch das Reverse Engineering per Definition *rückwärts*. Mit anderen Worten: Sie müssen in der Lage sein, das Pferd von hinten aufzuzäumen. Meditative Fähigkeiten zählen da weit mehr als viele Jahre formaler Programmierausbildung. Wenn Sie gut darin sind, Rätsel mit Ihren Freunden zu lösen, werden Sie wahrscheinlich auch gut im Reverse Engineering sein. Meister ihres Fachs wie +Fravia empfehlen das Reverse Engineering während der Einnahme stark alkoholhaltiger Getränke. Zwar können wir das aus gesundheitlichen Gründen natürlich nicht empfehlen, aber Sie werden feststel-



Abbildung 2-10: Der Installationsmanager In Control 5

len, dass auch eine Tasse heißen Tees den Kopf frei macht und das Rückwärtsdenken erleichtert. Die folgenden Abschnitte gehen einige reale Reverse-Engineering-Beispiele unter Windows durch.



Weil es illegal ist, die Schutzmechanismen urheberrechtlich geschützter Arbeiten zu umgehen, programmieren RCE-Experten zu Lehrzwecken mittlerweile eigene Schutzschemata. Diese so genannten *Crackmes* sind kleine Programme, die den Kern eines Schutzmechanismus und nur wenig sonst enthalten.

Beispiel 1: Ein Beispiel-Crackme

Beispiel 1 ist Muad'Dibs Crackme #1.



Die in diesem Kapitel vorgestellten Beispiel-Binaries (Crackmes) können von unserer Website *http://www.securitywarrior.com* heruntergeladen werden.

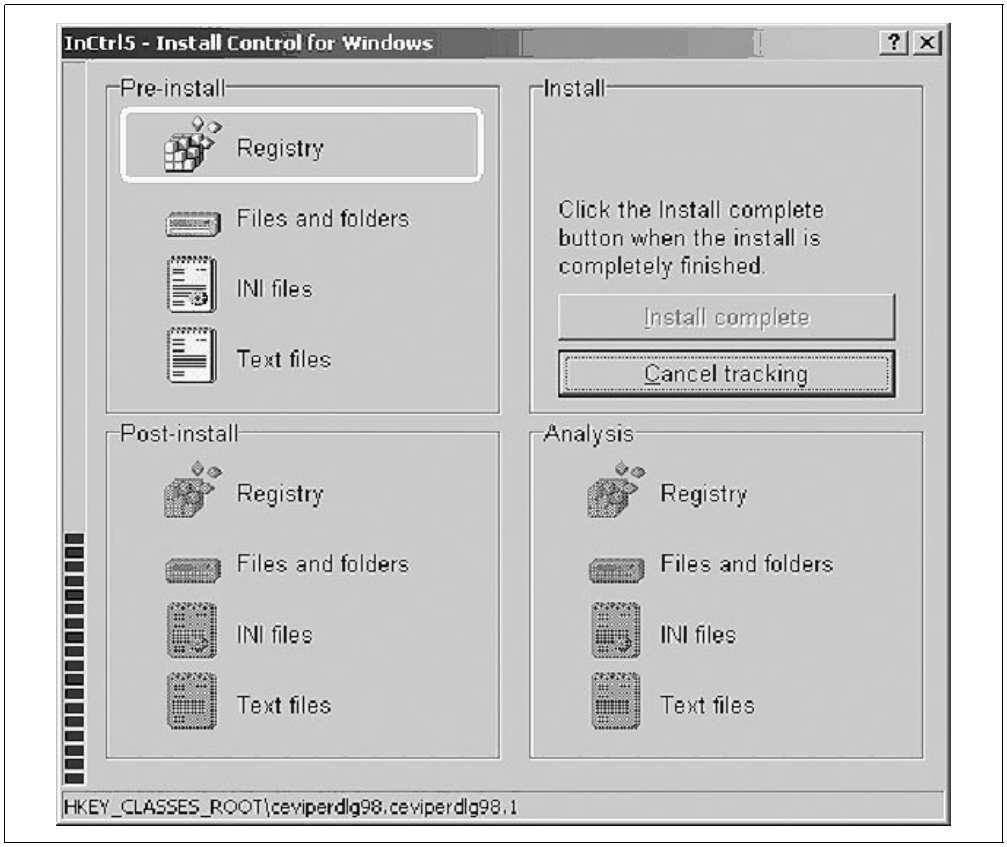


Abbildung 2-11: In Control 5 vergleicht Registry-Schlüssel, um herauszufinden, was installiert wurde

Dieses einfache Programm hat nur einen kleinen Haken. Die einzige Aufgabe des Programms besteht darin, uns davon abzuhalten, es zu schließen. Wenn Sie das Programm ausführen, sehen Sie einen Exit-Button, der aber (ganz bewusst) nicht funktioniert. Stattdessen erscheint ein ärgerliches Fenster, das Sie auffordert, den Exit-Button funktionsfähig zu machen (»Your job is to make me work as an exit button«, Abbildung 2-12).

Dieses Crackme verhält sich also wie Shareware oder Software, bei der bestimmte Features entfernt oder nur beschränkt nutzbar sind (so genannte *Crippleware*). Ihre Aufgabe besteht darin, das Programm voll funktionsfähig zu machen. Glücklicherweise liefert das Programm selbst einen wichtigen Hinweis. Indem man das disassemblierte Programm nach dem String

```
"Your job is to make me work as an exit button"
```

absucht, kann man den Sprung im Programm zurückverfolgen, der zur gewünschten Funktionalität führt – das heißt zu einem funktionierenden Exit-Button.



Abbildung 2-12: Muad'Dibs Crackme lösen

Nachdem Sie IDA Pro installiert haben, öffnen Sie die Zieldatei (in unserem Fall Muad'Dibs Crackme #1), warten auf die Disassemblierung und sehen dann den nackten Assembler-Code. Wir gehen den Schutzmechanismus direkt an, indem wir uns die praktische Liste von Strings ansehen, die IDA Pro für uns erzeugt hat (Abbildung 2-13).

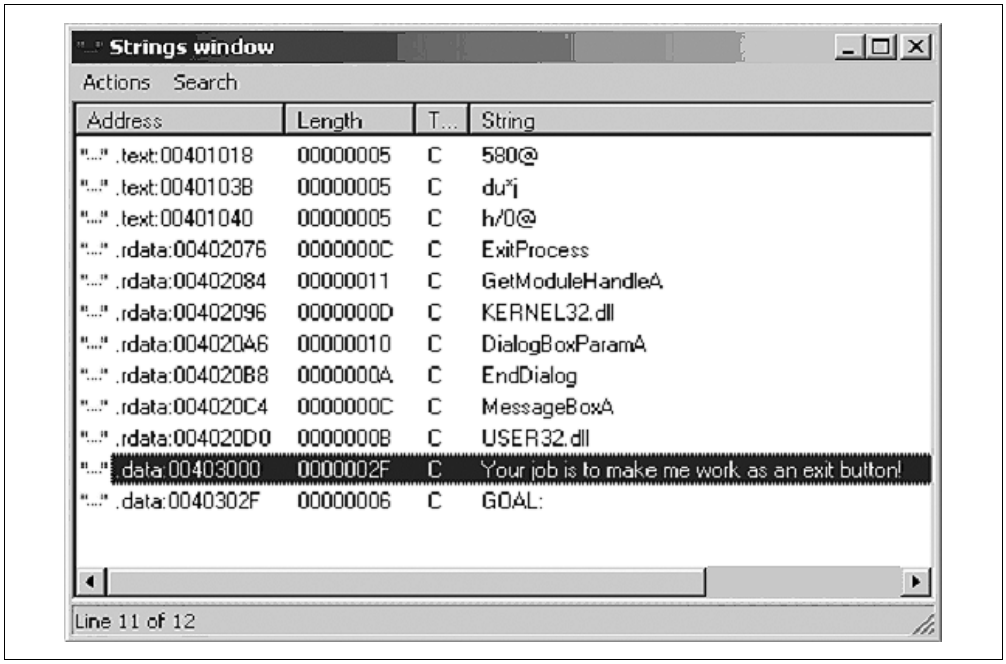


Abbildung 2-13: String-Disassembly in IDA Pro

Ein Doppelklick auf den gesuchten String bringt uns direkt zum gewünschten Code im disassemblierten Code (Abbildung 2-14).

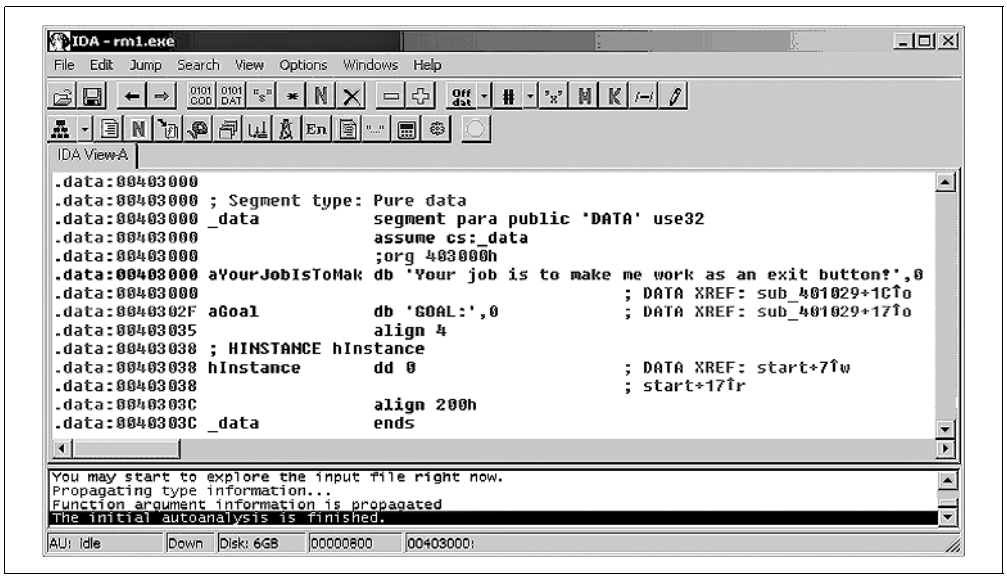


Abbildung 2-14: Den Zielcode im disassemblierten Code mit Hilfe von Strings finden

Wir landen bei folgendem Code:

```
* Reference To: KERNEL32.ExitProcess, Ord:0075h
                                   |
:00401024 E843000000               Call 0040106C
;( Ruft ExitProcess auf, wenn wir das Kreuz unter Windows anklicken)
:00401029 55                       push ebp
:0040102A 8BEC                     mov ebp, esp
:0040102C 817D0C11010000           cmp dword ptr [ebp+0C], 00000111
:00401033 751F                     jne 00401054
:00401035 8B4510                   mov eax, dword ptr [ebp+10]
:00401038 6683F864                 cmp ax, 0064
:0040103C 752A                     jne 00401068
:0040103E 6A00                     push 00000000

* Possible StringData Ref from Data Obj ->"GOAL:"
                                   |
:00401040 682F304000                   push 0040302F
; Hier wird der Text der MessageBox referenziert

* Possible StringData Ref from Data Obj ->"Your job is to make me work as an exit button!"
                                   |
:00401045 6800304000               push 00403000
:0040104A FF7508                   push [ebp+08]
; Diese Zeilen bringen die Überschrift und das Handle der MessageBox auf den Stack
```

```
* Reference To: USER32.MessageBoxA, Ord:01BBh

:0040104D E832000000                Call 00401080
:00401053 EB2A                      jmp 00401068
```

Dies ist der Aufruf der lästigen MessageBox, die wir umgehen wollen! Wir müssen diese Adresse so anpassen, dass zur API-Funktion *Exit Process* verzweigt wird. Dies ist der Kern des Schutzmechanismus.

Wenn wir uns die Zeile 401024 noch einmal genau ansehen, sehen wir, dass der Exit-Prozess 0040106C wie folgt aufgerufen wird:

```
* Referenced by a CALL at Address:
|:00401024 E843000000                Call 0040106C

* Reference To: KERNEL32.ExitProcess, Ord:0075h
; Dies ist der benötigte API-Aufruf von ExitProcess.
|:0040106C FF2504204000              jmp dword ptr [00402004]
```

Wir müssen also diesen Sprung korrigieren. Wir ersetzen die Bytes an Offset 40104D und 401053 durch die an Offset 40106C, und wenn wir dann den Exit-Button anklicken, wird das Programm ohne die lästige MessageBox beendet.

Am besten ersetzen Sie die Zeilen

```
:0040104D E832000000                Call 00401080
:00401053 EB2A                      jmp 00401068
```

durch

```
:0040104D FF2504204000              jmp dword ptr [00402004]
:00401053 90                        nop
```

Nun springt 0040104D an die ExitProcess-Adresse. Das Programm wird nun korrekt beendet, egal ob man das X oder den Exit-Button drückt. 00401053 ist nun überflüssig, weshalb wir es mit *NOP* auffüllen, das heißt, wir machen aus einem JMP ein NOP.

Für das eigentliche Opcode-Patching müssen wir unser Programm in einem Hex-Editor öffnen. Nachdem wir den Hex-Editor installiert haben, klicken wir das Binärprogramm einfach mit der rechten Maustaste an und wählen dann ÖFFNEN MIT → ULTRA EDIT. Es erscheint der reine Hexcode (Abbildung 2-15), den wir nun patchen können.

Wie finden wir die Bytes, die wir ändern müssen? Wir suchen den Hexdump nach einem eindeutigen String von Hex-Bytes ab, der den gewünschten Zielcode repräsentiert. Um beispielsweise

```
:0040104D E832000000                Call 00401080
:00401053 EB2A                      jmp 00401068
```

zu finden, suchen Sie nach diesem Hexstring (Abbildung 2-16):

```
E832000000EB2A
```

Der Schlüssel besteht darin, nach einem Hexstring zu suchen, der lang genug ist, um für die Anwendung eindeutig zu sein.

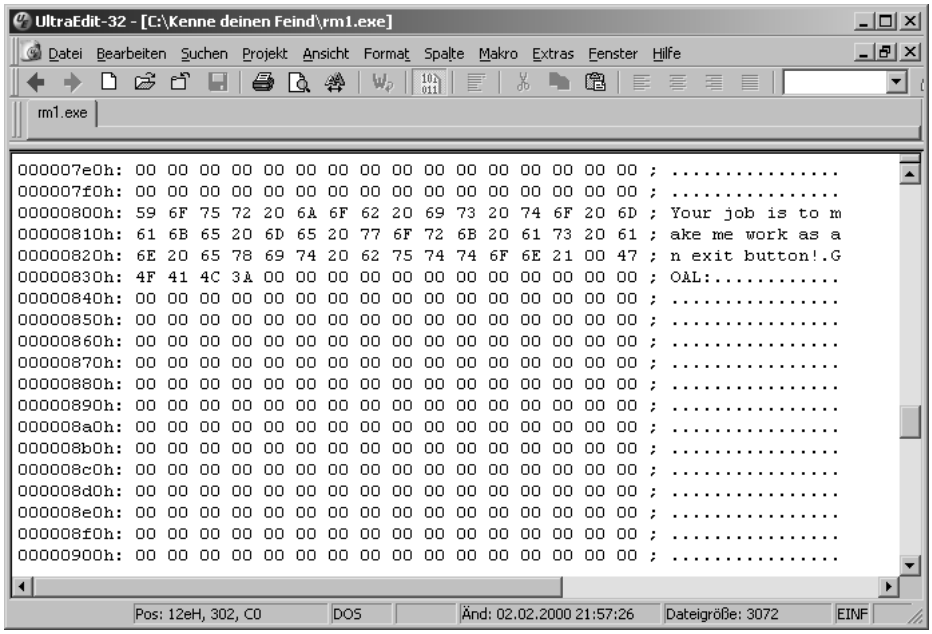


Abbildung 2-15: Hexdump unseres Binarys

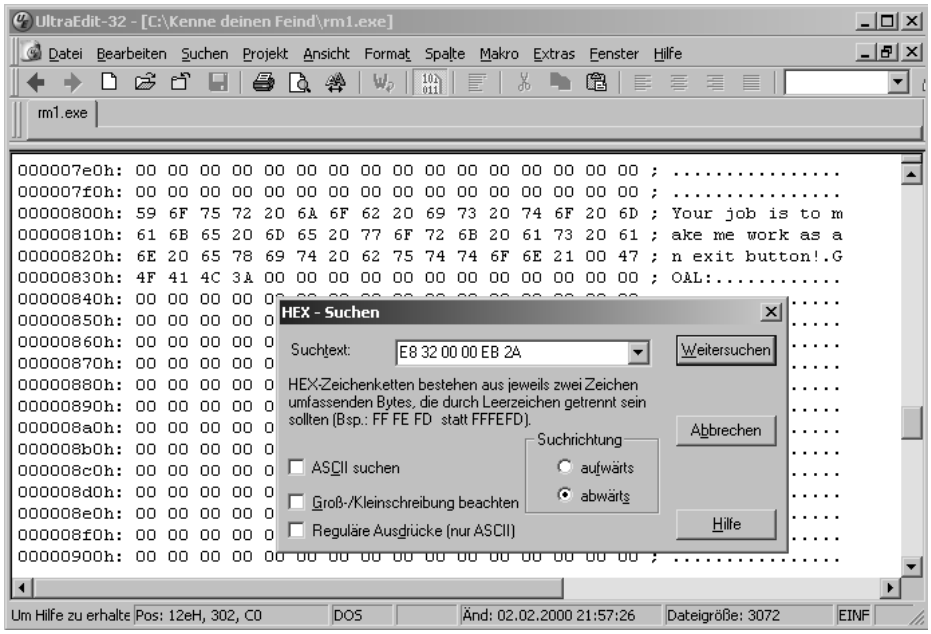


Abbildung 2-16: Suche nach dem zu korrigierenden Hexcode



Stellen Sie sicher, dass Sie nach Hex suchen und nicht nach ASCII.

Sobald Sie die Ziel-Bytes gefunden haben, ersetzen Sie diese vorsichtig, um den Sprung zu umgehen. Dann speichern Sie das Binary einfach ab und führen es erneut aus. In unserem Beispiel wird das Programm sauber beendet, wenn Sie den Exit-Button anklicken.

Beispiel 2: Bösartiger Code

Eine der wichtigsten Aufgaben des RCE ist das Reverse Engineering von bösartigen Codes wie Viren oder Trojanern. In diesem Beispiel wollen wir den lästigen SubSeven-Trojaner von MobMan untersuchen. Durch das Reverse Engineering eines Trojaners können Sie dessen eindeutige Hex-Signatur bestimmen, seine Registry-Einträge usw., die dann bei Antiviren-Programmen oder zur manuellen Isolation genutzt werden können. Allerdings wollen wir in diesem Beispiel SubSeven einem Reverse Engineering unterziehen, um Ihnen sein Geheimnis zu zeigen. Wir werden Ihnen zeigen, dass man heutzutage nicht einmal einem ehrlichen Trojaner-Entwickler vertrauen kann.

Während diese Zeilen geschrieben werden, können Sie sich den Trojaner von *http://www.subseven.ws* herunterladen. Sollte die Site irgendwann abgeschaltet werden (und das wird zweifellos passieren), suchen Sie einfach im Internet nach ihm. Die Ehre der Entdeckung des Geheimnisses geht an Defiler, und Teile werden mit Genehmigung von +Tsehp wiedergegeben. Für diese Übung benötigen Sie ein installiertes und funktionsfähiges SoftICE.

Sie können sich zwischen verschiedenen Versionen von SubSeven entscheiden, von denen jede leicht unterschiedliche Ergebnisse zurückliefert. Nach der Installation der Software konfigurieren Sie den Server mit dem beiliegenden Programm EditServer (Abbildung 2-17). In unserem Beispiel verwenden wir die localhost-Adresse für den Server und konfigurieren ihn mit Port 666 und dem Passwort »Peikari«.

Arbeiten Sie bei der Installation von Malware unbedingt mit einem Uninstall-Manager, damit Sie in der Lage sind, die Software später von Hand zu entfernen. Für die Übung müssen Sie Ihren Virenscanner deaktivieren, da Sie anderenfalls nicht in der Lage sind, die Software zu untersuchen. Sobald der Server konfiguriert ist, starten Sie den Client. Der erscheinende Disclaimer (Abbildung 2-18) ist recht ironisch, wie Sie gleich sehen werden.

Wir lassen den Client die Verbindung mit localhost (127.0.0.1) herstellen, wie in Abbildung 2-19 zu sehen. Beachten Sie, dass wir für den Port nicht die Voreinstellung 27374, sondern »666« verwenden (da wir den Server so konfiguriert haben).

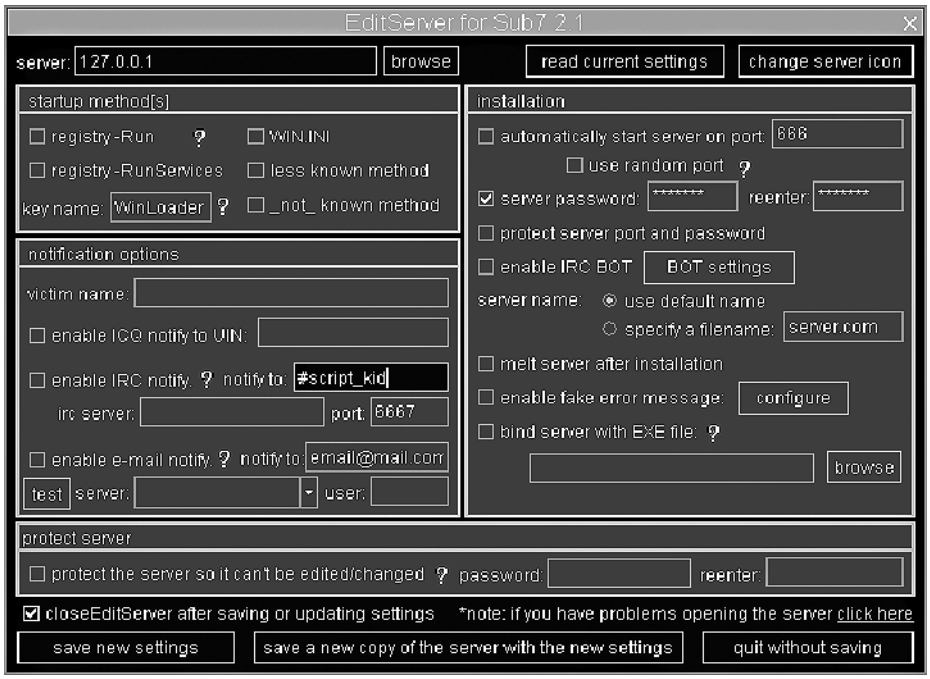


Abbildung 2-17: SubSeven mit EditServer konfigurieren

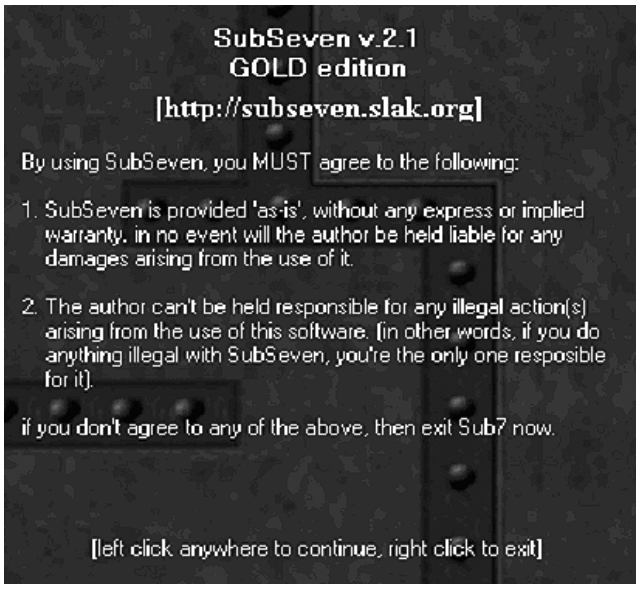


Abbildung 2-18: Der SubSeven-Disclaimer steckt voller ironischer Anspielungen

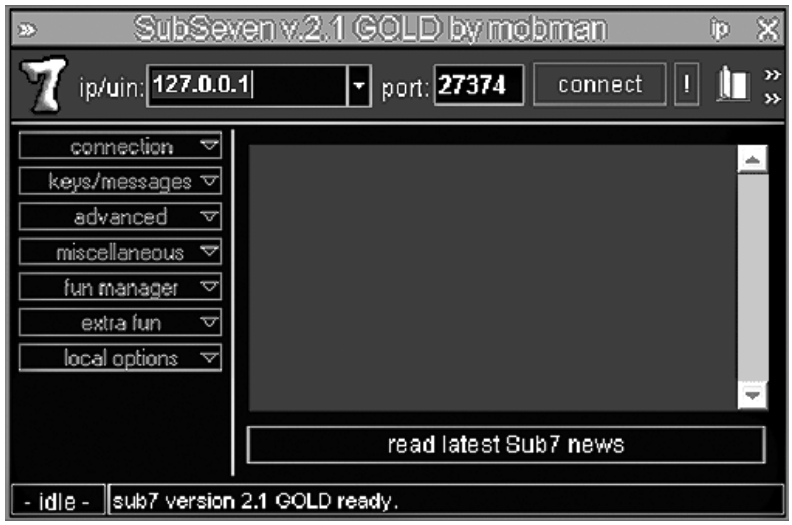


Abbildung 2-19: Mit dem SubSeven-Client die Verbindung zu localhost herstellen

Als Nächstes öffnen Sie den Symbol-Lader von SoftICE, um die Winsock-Exporte (*wsock32.dll*) zu importieren, die allerdings vom Betriebssystem abhängig sind. Nachdem Sie den SubSeven-Server in den Symbol-Lader von SoftICE übernommen haben, wird der Trojaner ausgeführt. Sobald Sie »connect« anklicken, um die Verbindung zu localhost herzustellen, erscheint der Passwort-Dialog. In diesem Fall geben wir ein Passwort ein, das sich von dem von uns gewählten Passwort (Peikari) unterscheidet.

Der Server verwendet die WSOCK32-Funktion recv, um die über das Socket gesendeten Daten zu empfangen:

```
int recv (

SOCKET s,
char FAR* buf,
int len,
int flags
);
```

Der zweite Parameter (`char FAR* buf`) ist der wichtige, da die Daten darin abgelegt werden. Bevor Sie das Passwort eingeben, drücken Sie Strg-D, um zu SoftICE zu wechseln. Setzen Sie nun wie folgt einen Breakpunkt für die recv-Funktion:

```
bpx recv do "d esp->8"
```

Drücken Sie nun erneut Strg-D und klicken Sie dann auf OK, um das Passwort an den Client zu senden. SoftICE wird am bpx anhalten. Drücken Sie F11 und Sie sehen Ihr Dummy-Passwort im SoftICE-Datenfenster zusammen mit dessen aktueller Adresse im Speicher.

Setzen Sie nun einen bpr für die Adresse des Passworts (zum Beispiel bpr 405000 405010 RW). Führen Sie das Programm erneut aus, und diesmal hält SoftICE das Programm an der Adresse 004040dd an. Sie sehen den folgenden Code:

```
0167:004040dd 8b0e        mov       ecx,[esi]  ; unser Passwort
0167:004040df 8b1f        mov       ebx,[edi]
0167:004040e1 39d9        cmp       ecx,ebx
0167:004040e3 755         jnz       0040413d
0167:004040e5 4a          dec       edx
0167:004040e6 7415        jz        004040fd
0167:004040e8 8b4e04      v         ecx,[esi+04] ; 4 Zeichen nach ecx
0167:004040eb 8b5f04      mov       ebx,[edi+04] ; 4 Zeichen des Master-Passworts nach ebx
0167:004040ee  39d9       cmp       ecx,ebx      ; beide Werte vergleichen
```

Das Programm wird in Zeile 4040dd angehalten, nachdem wir einen bpr für unser Dummy-Passwort gesetzt haben. Das bedeutet, dass das Passwort innerhalb des Puffers liegt, auf den esi verweist. Die ersten vier Zeichen werden nach ecx und vier Zeichen des Master-Passworts werden nach ebx verschoben. Diese werden dann miteinander verglichen.

Wir haben nun das cmp entdeckt, das unser Dummy-Passwort mit dem von uns gewählten vergleicht, nicht wahr? Falsch! Wir sind über die Tatsache gestolpert, dass der Autor von SubSeven *eine Hintertür in seine Hintertür eingebaut hat*! Geben Sie `d edi` ein, um sich die Daten des edi-Registers mit SoftICE anzusehen, und Sie werden Folgendes sehen:

```
016F:012A3DD4 31 34 34 33 38 31 33 36-37 38 32 37 31 35 31 30  1443813678271510
016F:012A3DE4 31 39 38 30 00 69 6F 00-28 00 00 00 22 00 00 00  1980.io.(..."...
016F:012A3DF4 01 00 00 00 13 00 00 00-53 75 62 73 65 76 65 6E  ........Subseven
016F:012A3E04 5F 5F 5F 3C 20 70 69 63-6B 20 3E 00 10 3E 2A 01  ___< pick >..>*.
016F:012A3E14 10 3E 2A 01 38 00 00 00-53 75 62 73 65 76 65 6E  .>*.8...Subseven
```

Diese Zahl (14438136782715101980) ist nicht das von uns gesetzte Passwort. Wir deaktivieren nun alle Breakpunkte (`bd *`), führen das Programm erneut aus und geben diesmal 14438136782715101980 als Passwort ein. SubSeven antwortet mit »connected«.

Dieses Beispiel zeigt, dass der Autor von SubSeven heimlich ein fest codiertes Master-Passwort in seine ganzen Trojaner eingebaut hat! Heutzutage kann man wirklich niemandem mehr vertrauen.

Referenzen

Die Beispiel-Crackmes dieses Kapitels finden Sie unter *http://www.securitywarrior.com*. Aufgrund ihrer kontroversen Natur besitzen einige der hier vorgestellten Referenzen häufig wechselnde URLs. Soweit möglich führen wir die aktualisierten Links unter *http://www.securitywarrior.com* auf.

- *Windows Internet Security: Protecting Your Critical Data* von Seth Fogie und Cyrus Peikari. Prentice Hall, 2001.
- ».NET Server Security: Architecture and Policy Vulnerabilities«. Eine auf der Defcon 10 im August 2002 vorgestellte Veröffentlichung.

- Die Tutorials zum PE-Format auf Iczelions Win32-Assembler-Website (*http://win32asm.cjb.net*)
- »Mammon_'s Tales to his Grandson: Mankind comes into the Ice Age«. Eine SoftICE-Einführung.
- »Mammon_'s Tales to Fravia's Grandson: An IDA Primer«
- SoftICE-Breakpunkte (*http://www.anticrack.de*)
- »WoRKiNG WiTH UCF's ProcDump32« von Hades
- *Win32 Assembly Tutorial*. Copyright 2000 by Exagone. (*http://exagone.cjb.net*)
- Offizielle SubSeven-Site (*http://www.subseven.ws*)
- »Trojan Reversing part I - by defiler.« Veröffentlicht von +Tsehp, September 2000
- Muad'Dibs Crackme, veröffentlicht von +Tsehp