=

Cyber Security Labor (cysL)

Martin Gwerder

..

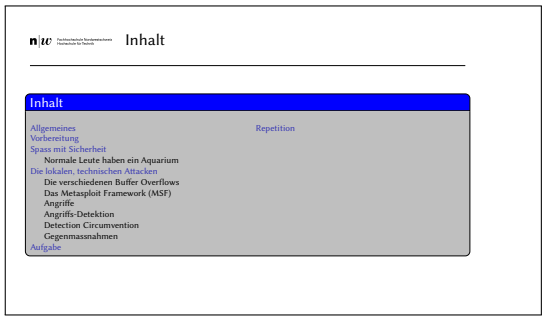

Abbildung 1: Inhaltsverzeichnis

Inhaltsverzeichnis

1 Allgemeines

2 Vorbereitung

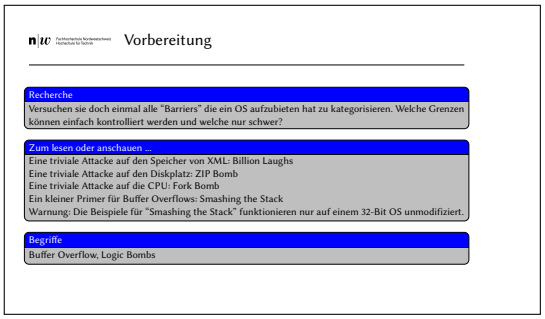

Abbildung 2: Folie "Vorbereitung"/""

Bitte arbeiten sie jeweils die Vorbereitungs-Slide vor(!) einer Lektion durch. Sie erlauben es damit, dass alle mit einem bestimmten Vorwissen in die Lektion kommen. Dies führt zu interessanteren, abwechslungsreicheren Diskussionen.

3 Spass mit Sicherheit

3.1 Normale Leute haben ein Aquarium

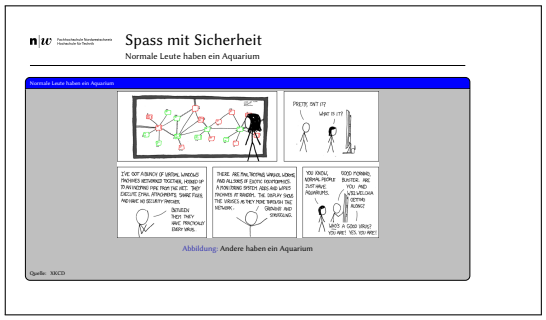

Abbildung 4: Folie “Spass mit Sicherheit”/“Normale Leute haben ein Aquarium”

Manche Leute haben ein Aquarium und andere pflegen Viren. Wichtig ist aber nicht was sie haben sondern wie sie damit umgehen. Überlegen sie sich doch einmal wie “gefährliche” Programme gehandhabt werden müssen. Sie benötigen dafür normalerweise ein “Aquarium”. Dieses Aquarium benötigt alles was sie brauchen um das Programm am “leben” zu erhalten. Ausserdem muss es frei von schädlichen Einflüssen sein. Ausgerechnet ein Virenscanner oder eine Malware-Detection hat in einem solchen Umfeld nichts (oder vielleicht bloss besuchsweise) etwas zu suchen. Dafür benötigen sie für eine DriveBy-Infektion einen Webbrowser und einen Webserver um sie “laufen zu lassen” und eine Möglichkeit den Exploit wieder “einzufangen”.

4 Die lokalen, technischen Attacken

4.1 Die verschiedenen Buffer Overflows

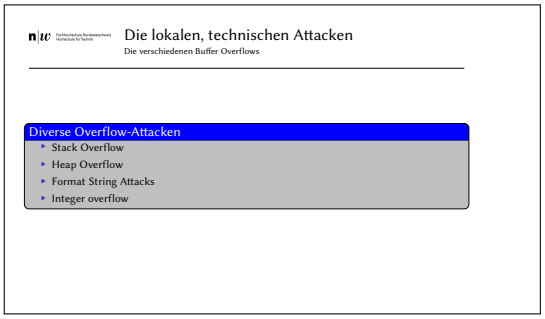

Abbildung 5: Folie “Die lokalen, technischen Attacken”/“Die verschiedenen Buffer Overflows”

- Stack Overflow
  Dies ist ein klassischer Overflow. Viele Lowlevel-Programme sind dafür anfällig, weil der Stack sehr knapp bemessen ist.
- Heap Overflow
  Eigentlich analog zum Stack Overflow. Im Gegensatz zum Stack ist der Heap normalerweise grosszügiger bemessen.
- Format String Attacks
  Format Attacken erfolgen, wenn es beim %n-Parameter (n ist eine Zahl) möglich wird ausserhalb den Zielbuffer zu überfüllen. Dies ist üblicherweise eine Sonderform des Stack-Overflows.
- Integer overflow
  Integer sind häufig vorzeichenbehaftet in ihrer Interpretation, so dass $max + 1 = -max + 1$ ergibt. Diesen Umstand kann man für einen Angriff ausnutzen. Dies vor allem, wenn Operationen auf einer Eingabe ausgeführt werden.

Es gibt aber auch noch viele unkonventionellere Overflowattacken:

- Memory exhaustion
  Speicher vom Speichermanager anfordern bis kritische Codeteile (typisch: Treiber) gepaged werden. Anschliessend werden die Codeteile direkt auf der Disk ausgetauscht und verwendet.
- Hibernation Attack
  Im offline Modus Modifikationen am Hibernation-File vornehmen und anschliessend regulär booten.

4.2 Das Metasploit Framework (MSF)

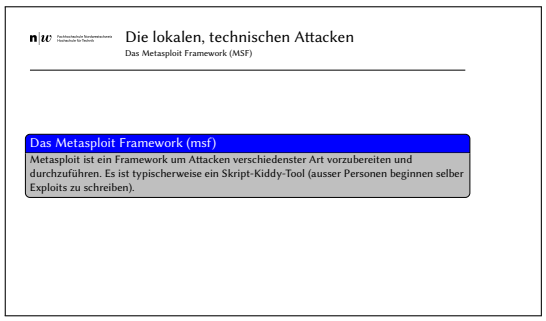

Abbildung 6: Folie "Die lokalen, technischen Attacken"/"Das Metasploit Framework (MSF)"

Erste Schritte bei Kali

- Kali updaten (`apt-get update && apt-get upgrade`)
- Dienste (neu-)starten (`service postgres restart && service metasploit restart`)
- Metasploit-DB updaten (`msfupdate`)

Metasploit ist ein Framework um Attacken verschiedenster Art vorzubereiten und durchzuführen. Die Methodologie richtet sich nach folgendem Schema:

- Identify Target
- Scan Target
- Exploit/Payload selection
- Exploit configuration
- Payload configuration
- Exploit & Post-Exploit

Ein Beispiel finden sie im Metasploit Primer (http://www.isaca.org/...Metasploit%20Primer.pdf)

4.3 Angriffe

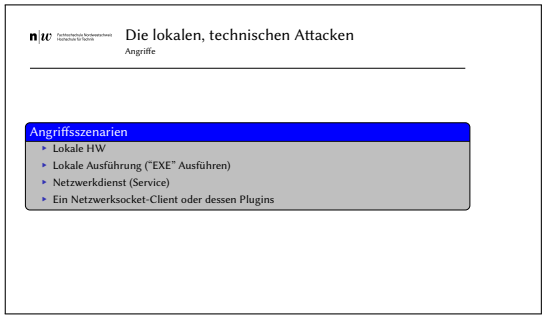

Abbildung 7: Folie "Die lokalen, technischen Attacken"/"Angriffe"

Verschiedene Angriffsszenarien sind normalerweise verfügbar. Natürlich sind Angriffe auf ein Netzwerkdienst am attaraktivsten, weil man diese problemlos jederzeit ausführen kann. Glücklicherweise werden diese in den bekannteren Programmen/OSes mittlerweile zeitnah ausgemerzt. Es gibt aber noch andere Angriffsvektoren:

- Lokale HW
  Normalerweise wird lokal angeschlossener HW blind vertraut.
- Lokale Ausführung ("EXE" Ausführen)
  Meistens ist es kein Problem aus einer lokalen Einschränkung (Sandbox, Jail oder User-Rechte) auszubrechen, sobald man ausgeführten Code auf einem System hat. Das lässt sich am besten mit Metasploit zeigen → Auf das System kommen ist schwierig. Nachher reicht üblicherweise ein `get system` um die erhöhten Privilegien auszukosten.
- Netzwerkdienst (Service)
  Das attraktivste Angriffsziel…"Immer und überall nur zugreifen"
- Ein Netzwerksocket-Client oder dessen Plugins
  Primäres Ziel für viele Phishing Angriffe. Werden häufig als Drive-by Infektionen realisiert.

4.4 Angriffs-Detektion

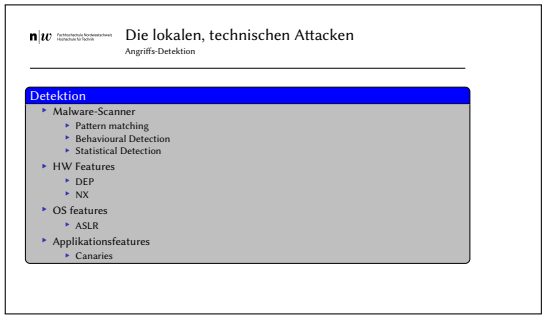

Abbildung 8: Folie "Die lokalen, technischen Attacken"/"Angriffs-Detektion"

Es gibt viele Verteidigungslinien bei den Overflows. Die Klassischen seien hier aufgezählt:

- Malware-Scanner
  Typischerweise ein "on access"-Scanner. Er Scannt die Files, sobald sie von der Disk geladen werden. Zunehmend gilt dies auch für den zugriff auf unverschlüsselte Seiten. oder auch via bekannter Highlevel-APIs. Dies ist aber sehr einfach zu Umgehen.
  - Pattern matching
    Erkennt gefährliche Programme anhand von definierten Mustern (kann trivial sein wie Regex aber es werden zunehmend auch Algorithmen aus der Bioinformatik eingesetzt), die an definierten Stellen gesucht werden.
  - Behavioural Detection
    Diese Detektion erfolgt aufgrund verdächtiger Aktivitäten (z.B. ändert ausführbare Dateien auf der Disk, lädt Files von bekannten (schlechten) URLs).
  - Statistical Detection
    Codeteile beginnen viral (an mehreren Orten gleichzeitig) an aufzutauchen.
- HW Features
  Immer mehr wird sicherheit zur "Chefsache". Die Hardware selbst kennt heute verschiedene Barrieren.
  - TPM-Module
    Das Trusted Platform Module stellt sicher, das nur "vertrauenswürdiger" Code als Basis zur Ausführung kommt.
  - DEP/ESP/NX
    Bei dieser Technologie geht es darum Codeteile, die als Daten geladen wurden, als nicht ausführbar zu markieren. Werden sie dennoch ausgeführt gibt es eine Exception.
- OS features
  - ASLR
    Dies ist keine direkte Abwehr. Adress Space Layout Randomisation sorgt für ein nicht vorhersehbares Memory-Layout. Die Technik macht Buffer Overflows nicht unmöglich aber die Effekte sind schwerer zu reproduzieren. Gegen diese generische Abwehr gibt es mit Heap-Spraying aber bereits eine genau so generische Circumvention-Methode.
- Applikationsfeatures

- Canaries
  Canaries sind typischerweise durch Compiler in Programmen implementiert. Seltenerweise auch vom OS. Bei Canaries werden Variablen mit zufälligem Inhalt auf den Heap (oder den Stack) geschoben. Werden sie überschrieben liegt ein Buffer Overflow vor.

4.5 Detection Circumvention

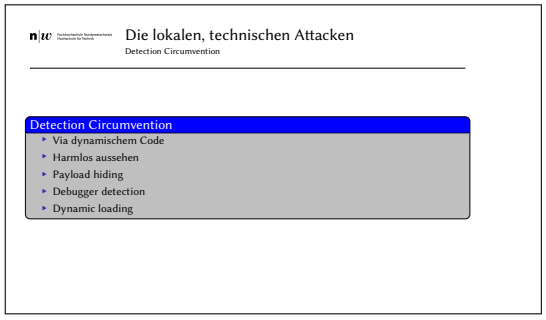

Abbildung 9: Folie “Die lokalen, technischen Attacken”/“Detection Circumvention”

Bei der Detection Circumvention geht es darum, die normalen detektoren auszutricksen. Anbei ein paar der beliebtesten Tricks, wenn es darum geht Virenscanner zu umgehen:

- Via dynamischem Code
  Wenn der Code sich selber umschreibt (kann dauernd oder in Generationen erfolgen) ist es schwierig für einen Scanner ihn wiederzuerkennen. Am einfachsten wird dies durch einfügen von “NOP”s erreicht. Es gibt aber auch kompliziertere Algorithmen, die Codeblöcke gegeneinander verschieben oder die Code-Reihenfolge umstellen kann. Neuartige Construction-Kits verwenden Informationen des Compilers um den Code umzustellen. Diese Code Reorderer werden häufig “zuunterst” im Code versteckt.
- Harmlos aussehen
  Man kann Schadcode verstecken, indem man ihn in viel Codemüll versteckt. Virenscanner scannen typischerweise nicht ganze Files sondern nur interessante Codeteile (aus Effizienzgründen).
- Payload hiding (Laufzeitverschlüsselung oder Laufzeitmodifikation)
  Im einfachsten Fall ist dies eine Verschlüsselung oder eine XOR-Verschleierung. Dies sorgt dafür, das kritische Codeteile nicht von einem Patternscanner auf der Disk entdeckt werden können.
- Debugger Detection
  Codeteile können sich anders verhalten, wenn sie davon ausgehen, dass ein Code-Debugger mitliest. Das macht die Analyse des Codes extrem schwer.
- Dynamic loading
  Codeteile können nachgeladen werden. Vorzugsweise zur Laufzeit, dynamisch (nicht über Link Tables) und direkt von alternativen Quellen (Internet). Das umgeht den “on access”-Scanner auf der Disk.

4.6 Gegenmassnahmen

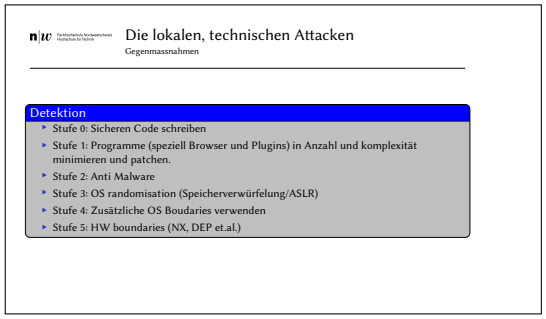

Abbildung 10: Folie “Die lokalen, technischen Attacken”/“Gegenmassnahmen”

Wir kennen mehrere Stufen bei der Abwehr:

- Stufe 0: Sicheren Code schreiben
  Das wäre die wichtigste Form der Abwehr. Leider sind praktisch alle Programmierer dafür bezahlt Features zu implementieren. Sicherheit spielt, wenn überhaupt, immer eine absolut untergeordnete Rolle.
- Stufe 1: Programme (speziell Browser und Plugins) in Anzahl und komplexität minimieren und patchen.
  Das ist der Versuch, den viele Sicherheitsbeauftragte in Firmen versuchen durchzusetzen. Das Problem ist, dass heute das Verwenden von Module und “Buildingblocks” zum Alltag gehört und sich immer Gründe dafür finden lassen, warum bestehende Lösungen nicht ausreichen.
- Stufe 2: Anti Malware (am besten mehrere aber nicht auf der selben Maschine)
  Es ist heute üblich in grossen Firmen mindestens zwei verschiedene Malware-Scanner im Einsatz zu haben. Meistens einer auf Servern und ein anderer auf Clients.
- Stufe 3: OS randomisation (Speicherverwürfelung/ASLR)
  Diese Verteidigung ist heute vielerorts standardmässig vorhanden. Ebenso standardmässig wird heute gesprayt.
- Stufe 4: Zusätzliche OS Boudaries verwenden
  Hier seien Dinge wie Canaries, Jails, Sandboxes und ähnliches erwähnt. Rechte entfernen bringt normalerweise nichts oder nur wenig, weil wenn man erst auf einem System ist die Privilegien eigentlich immer eskaliert werden können.
- Stufe 5: HW boundaries (NX, DEP et.al.)
  Diese werden heute zunehmend automatisch eingesetzt. Viele Modernen Betriebssysteme verlangen bereits standardmässig, das neue Programme diese Features unterstützen müssen.

5 Aufgabe

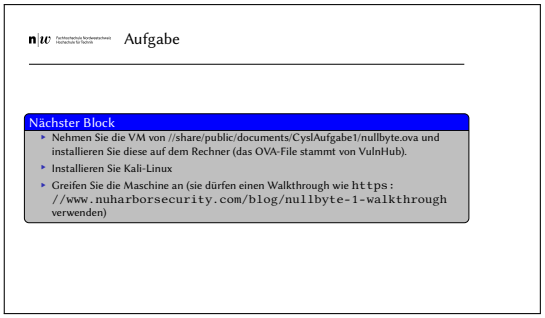

Abbildung 11: Folie “Aufgabe”/“”

- Nehmen Sie die VM von //share/public/documents/CyslAufgabe1/nullbyte.ova und installieren Sie diese auf dem Rechner (das OVA-File stammt von VulnHub1).
- Installieren Sie Kali-Linux
- Greifen Sie die Maschine an (sie dürfen einen Walkthrough wie `https://www.nuharborsecurity.com/blog/nullbyte-1-walkthrough` verwenden)

6 Repetition

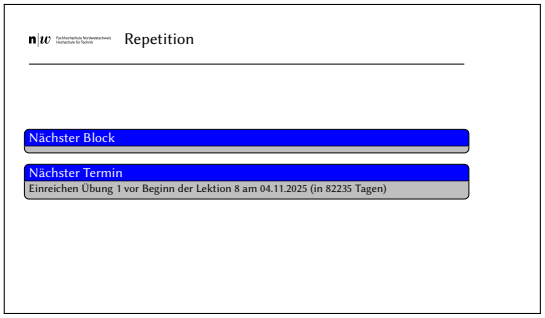

Abbildung 12: Folie “Repetition”/“”