1. Klausurvorberitung

Sonntag, 8. Februar 2015 16:06

- Übergang von verschiedenen Sicherheitsstufen als Sicherheitsrisiko
- Race conditions und TOCTOU
- Konzepte vom Systemaufruf, Speicherschutz und Zugriffskontrolle
- Speicherlayout
- Funktionsaufrufe
- Buffer overflow
  - Erkennen von Porblemen in Quelltexten
  - Angriffe erklären
  - Verteidigung gegen Angriffe
- Schwachstellen
  - format string
  - return-into-libc
- Klassifizierung von Malware und Aufbau von Standard Malwaretypen
- Konzept hinter Bots und Botnetzen
- Malwareverteidigung
- Konzept hinter confinement
- Aho–Corasick Algorithmus
- Konzept von Honeypots und Honeynets

2. Zusammenfassung

Sonntag, 8. Februar 2015 16:20

Systemaufrufe

Hardware interrupts

-

Speicherschutz

- Jeder Prozess hat eigenen virtuellen Speicher
- Es gibt keinen Zugriff auf den physischen Speicher
- Page tables und MMU sind für die Speicherverwaltung zuständig

Zugriffskontrolle

- Bestimmt die Aktion, welche ein Prozess auf ein Objekt ausüben kann
- Erstellt eine Identität für ein Subjekt (Owner)
- Fügt ACL (accesss control lists) zu einem objekt hinzu

Uneingeschränkte Zugriffskontrolle

- Gängiges Vorgehensweise bei heutigen Systemen
- Subjekt kann Berechtigung auf ein Objekt ändern

Zwangsweise Zugriffskontrolle

- Nicht so gängige Vorgehensweise
- Erzwungen durch OS, falls das Subjekt die Berechtigungen nicht ändern kann
- Meist verbunden mit MLS (multi-level security) und Bell-LaPadula

3. Unix/Linux Sicherheit

Sonntag, 8. Februar 2015 16:41

Benutzer

- Aufgeteilt in user name (UID) und group name (GID)
- Meist du ein Passwort geschützt
- Passwort wird meist mit Salt gehasht, damit Rainbowtables und andere angriffe schwieriger sind
- Passwortdatei wird benötigt, da so eine ID einem Nutzer zugewiesen wird

Root-Nutzer

- Auf superuser oder system administrator genannt
- Besondere Berechtigungen

Gruppen

- groupname : password : group id : additional users
- Ein Benutzer gehört zu einer oder mehreren Gruppen
- Hauptgruppen liegen in der /etc/password
- Zusatzgruppen liegen in /etc/group

Process vs. Thread

- Process
    - Beinhaltet Nutzeraktivitäten
    - Einheit, welche einen bestimmten Code ausführt
    - Hat einen eigenen Stack, File Descriptor Tables und Speicherseiten
    - Durch virtuellen Speicher von anderen Prozessen getrennt
- Thread
    - Ein Prozess kann mehrere Threads haben
    - Threads haben eigenen Stack und Programm Counter
    - Teilt sich aber Speicherseiten und File Descriptor Tables

Prozessaufbau

- Process ID(PID) kann den Porzess klar identifizieren
- Real user ID(UID) Porzess kann einem bestimmten benutzer zugeordnet werden
- Effective user ID(EUID) wird zum Zugriffscheck benötigt
- Saved user ID(SUID) wird temporär für Berechtigungen benötigt
- Jeder Prozess hat eine reelle und eine effektive Benutzer-/Gruppen-ID
- Reelle ID wird typischerweise durch aktuellen Nutzer mit Authentifizierung gesetzt
- Effektive ID wird zur Rechtevergabe von Prozessen durch das System gesetzt

4. Dateisysteme und Race Conditions

Sonntag, 8. Februar 2015 17:10

Dateibaum

- Primärer Ablageort von Informationen
- Hierarchisches Ablagesystem
- Verzeichnisse bestehen aus File System Objects (FSO)
- Das Wurzelverzeichnis ist "/"

FSO

- Ermöglicht das verarbeiten und behandeln von Ordnern und Dateien

Zugriffskontrolle

- Berechtigungsbits für Datei
- Chmod, chown, chgrp, umask
- user-rwx, group-rwx, other-rwx
- read(r), write(w), execute(x), suid/sgid(s), files only deletable by owner(t)

Race Condition

- Geteilte Objekte sind anfällig für Race Condition Probleme
- Time-of-Check, Time-of-Use (**TOCTOU**) Angriff
    - Time-of-Check (t1): kontrolliert Voraussetzung von A auf das Objekt
    - Time-of-Use (t2): Angnommen A hat die Voraussetzungen, kann das Objekt benutzt werden
    - Time-of-Attack (t3): Angenommen A ist nicht valide
    - Angriff t1 < t3 < t2

TOCTOU

- Schritte des Zugriffes auf ein Objekt
    - Bezieht Informationen über das Objekt
    - Abfrage des Objektes auf Eigenschaften
    - Auswertung der Anfrage
    - Falls das Objekt zulässig ist, wird darauf zugegriffen
- Programm wird überprüft
- Programm ist ein Script -> Interpreter wird aufgerufen
- ANGRIFF: Angreifer ändert das Script
- Interpreter mit root-Rechten führt Script aus

Shell

- Die Shell ist eine Kern Anwendung von Unix Systemen
- Sie besitzt eine Kommando- und eine Programmiersprache
- Viele Wichtige Funktionen können mit der Shell benutzt werden
- Leichte Eingabe mit vielen Parametern

Shell Angriffe

- Kontrollieren und Verlassen von Funktionen
- Befehle der Shell können in einfachen Strings untergebracht werden

Shellshock

- Kommando wird ausgeführt wenn es mit dem Ende von Umgebungsvariablen verkettet ist

5. Windows Security

Sonntag, 8. Februar 2015 18:34

Security Reference Monitor (SRM)

- Ist ein Kernel Prozesse
- Entscheidet über die Zugriffskontrolle
- Erstellt die Zugriffsrichtlinien

Local Security Authentication (LSA)

- Ist ein Benutzer Prozess
- Verwaltet die Zugriffsberechtigungen
- Managed die Benutzer Authentifizierung
- Gibt user SID und group SID aus

Windows Logon

- Ist ein Benutzer Prozess
- Sammelt die Login-Informationen

Tokens

- Vergleichbar mit UID und GID für UNIX Prozesse
- Folgeprozesse erben Zugriffstokens
- Unterschiedliche Folgeprozesse können unterschiedliche rechte haben
- Ein Prozess kann den Token erstellen und vererben

Zugriffskontrollentscheidungen

Objekt

- Windows ist objektorientiert
- Alles ist ein Objekt
- Jedes Objekt hat Sicherheitseinstellungen

Subjekt

- Threads und Prozesse
- Haben einen Sicherheitsumgebung

Operation

- Entscheidet über die Zugriffsberechtigung (read, wirte, delete, …)
- Falls die Berechtigung vorliegt wird bei einem Aufruf ein Objekt handle zurück gegeben

Sicherheitsumgebung

- Wird in Zugriffstokens gespeichert
- Verbunden mit jedem Thread oder Prozess
- Zugriffstoken
    - User SID (Security IDentifiers)
    - Group SIDs
    - Rechte
    - Standard Rechte für angelegte Dateien
    - Management Informationen

Dateisysteme

- File Allocation Table (FAT)
- NT File System (NTFS)

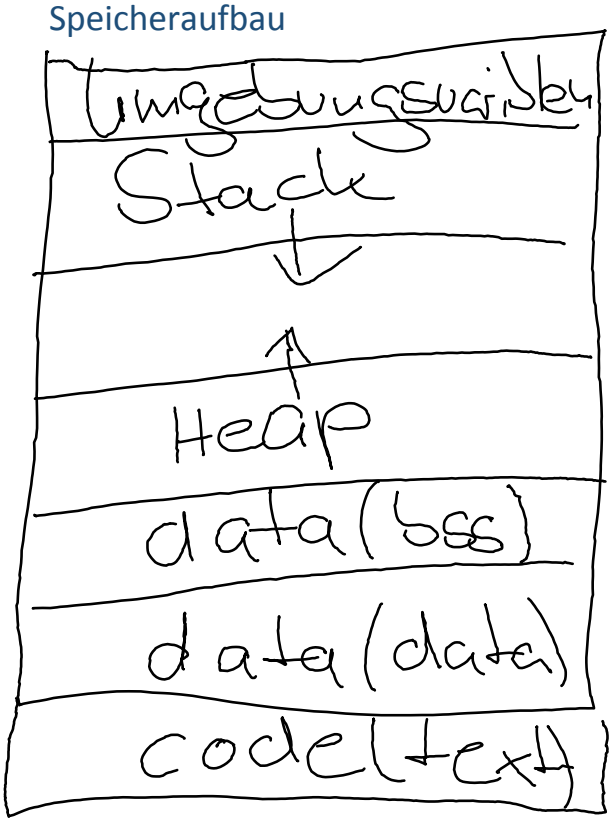
Verschlüsselung

- BitLocker und BitLocker 2 go
- Encrypted File System (EFS)

6. X86 Basics

Montag, 9. Februar 2015 16:53

Speicheraufbau

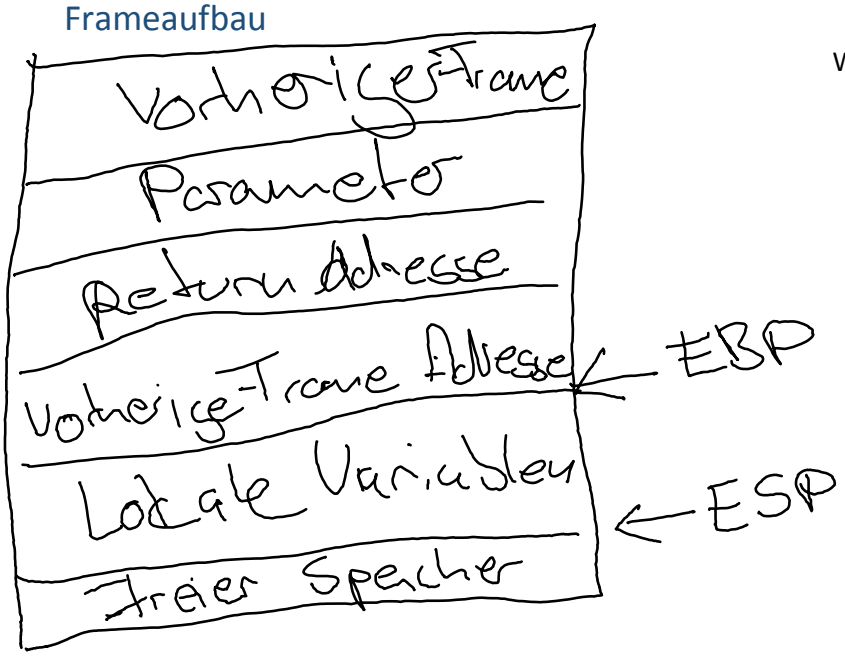


Stackabschnitt:
- Lokale Variablen

Datenabschnitt:
- Globale uninitialisierte Variablen (bss)
- Globale initialisierte Variablen (data)
- Dynamische Variablen (heap)

Codeabschnitt:
- Programminstruktionen
- Normalerweise nur lesbar

Frameaufbau



Wichtige Register
- EAX = Akkumulator (Speicher)
- EBX = Basis
- ECX = Zähler
- EDX = Data
- ESI = Quelle (Source)
- EDI = Ziel (Destination)
- ESP = Stack Pointer
- EBP = Base Pointer
- EIP = Instruction Pointer
- EFLAGS = Status Flag

7. Buffer Overflow

Montag, 9. Februar 2015 17:10

Vorgehensweise

- Man muss einen Pointer finden, welcher eine Sprungadresse enthält
- Man muss einen Overflow erzeugen, dass die Adresse seines Codes in den Pointer geschrieben wird

Gegenmaßnahmen

- Stack Canaries
  - Zwischen old ebp und Code wird ein Wert im Stack gespeichert
  - Bevor man zurück springt, wird dieser Wert überprüft
  - Falls dieser nicht der eingetragenen ist, so ist dies ein Buffer Overflow
- Non-Excutable Stack
  - Moderne CPUs/OSs können bestimmte verhindern, dass bestimmte Speicherseiten nicht ausgeführt werden
  - Data Execution Protection DEP
- Adress-Space Layout Randomization
  - Bei jedem Aufruf des Programmes, hat der Speicher eine andere Größe
  - Shared libraries und Code sind immer an anderen Positionen
  - Macht es schwer eine Adresse zu raten
  - Kann mit BruteForce gebrochen werden

Achten auf folgende Funktionen

a. Strcpy
b. Strcat
c. Sprintf
d. Char*e

8. Confinement Problem

Montag, 9. Februar 2015 18:38

Goal of service provider

- Der Server muss sich sicher sein, dass die Ressource, welche er nutz dem User gehört und sicher ist

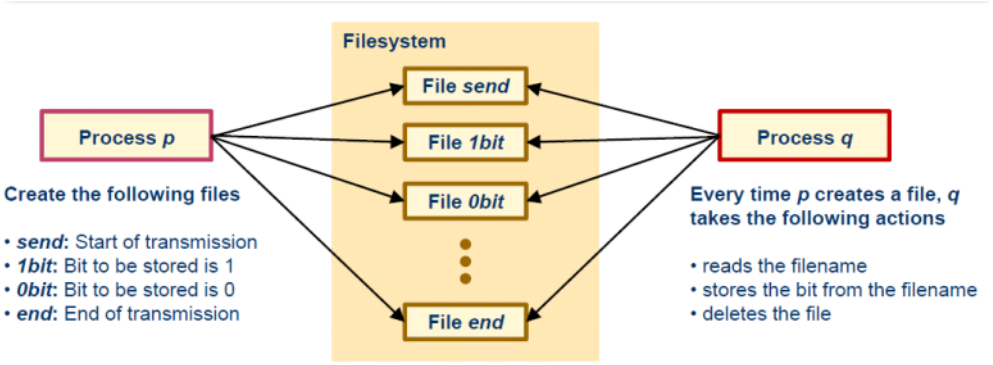
Goal of service user

- Der Nutzer muss sich sicher sein, dass niemand außer dem gewählte Server seine Daten lesen kann

Ein Prozess muss sich sicher sein, dann kein Prozess den er nutzt die gegebenen Informationen weiter gibt

Covert Channels

Resources designed for communication (pipes, shared memory)
Resources not designed for communication (storage or CPU) <- covert channel



This is Covert storage channel

Noiseless vs Noisy

Noiseless: the shared resources only shared with reciever and sender
Noisy: more the the reciever and the sender have access to the shared resource

Eigenschaften

- Existenz: es existiert ein Kanal über den Informationen übertragen werden können
- Bandbreite: Geschwindigkeit der Informationsübertragung

Ziele der Covert Channel Analyse

Entdecken und Zerstören
Falls Zerstören nicht möglich ist, wird die Bandbreite minimiert

Covert Channel Detection

- Shared Resource Matrix (SRM) methodology
  - Alle Shared resources mit ihren Attributen (read, modify)
- Information flow analysis
  1. Identifiziere Schlüsselfunktionen (wichtige Funktionen mit hohen Berechtigungen)
  2. Identifiziere Schlüsselvariablen
  3. Analyse von Variablen welche mit anderen Prozessen verbunden sind
- Non-interference
  - Falls ein Prozess mit einem anderen Prozess interferieren kann existiert ein Covert Channel
  - Alle Prozesse einer Ebene haben dieselben Informationen

- Höhere Prozesse können die Informationen von niedrigen Prozessen nicht beeinflussen

Analyse

- Wie gefährlich sind Covert Channels
- Wie groß ist die Bandbreite?

Gegenmaßnahmen (Mitigation)

- Zerstören ist sehr schwierig
- Isolation eines Prozesses
- Zufälligkeit einbauen

Isolation

Virtuelle Maschine

Der Prozess wird in einem isolierten System ausgeführt
Ein Computer wird simuliert
Der Prozess hat nicht die Möglichkeit auf das Hostsystem zuzugreifen
Side und Covert Channels existieren noch
Beispiel Java Virtual Machine (JVM)

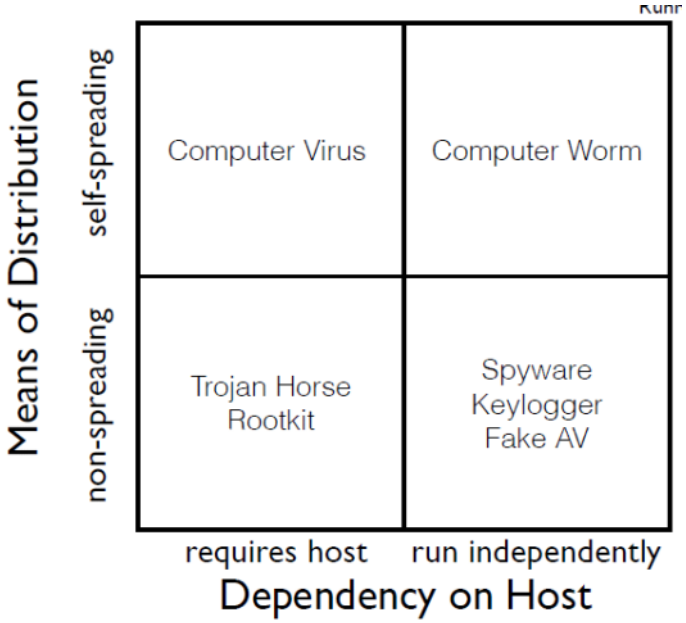
Sandbox

Der Prozess wird bei der Laufzeit analysiert und davon abgehalten Informationen weiter zu geben
Es wird kein eigener Computer simuliert
Die Prozesse sind beschränkt in ihrer Ausführung

- Es werden extra Sicherheitsmaßnahmen in Libraries oder Kernel geladen
  - Porgramm wird nicht verändert
- Das Programm wird in der Laufzeit verändert
  - Durch Debugger werden an wichtigen Stellen Breakpoint eingesetzt und der Status des Prozesses wird überprüft

9. Malware (Malicious Software)

Dienstag, 10. Februar 2015 10:55



Virus

Ein Virus verbreitet seinen Code, indem er seinen Code in andere executable files lädt und sich so reproduziert

1. Reproduce
2. Infect
    a. Der Virus versucht zu infizieren ohne aufzufallen
    b. Keine Vorhersage möglich
    c. Vereint Viruscode mit Programmcode
3. Run Payload / Attack
    a. Dateien löschen oder Daten auf der Festplatte verändern
    b. Viren haben häufig Fehler, so kommt es meist zu Schäden

Worm

Verbreitet sich selbstständig über ein Netzwerk

- *Interaction-based worms*
    - Need human interaction
    - Double-Click
    - Follow link
- *Process-based worms*
    - Netzwerkservices werden infiziert

1. Target Locator
    a. Email, Shared Files, Googlesearch
2. Infection Propagtor
    a. Kontrolle über das Opfer erlangen
    b. Worm Body einschleusen
3. Life Cycle Manager
    a. Zeitabhängig verändert sich sein verhalten
4. Payload
    a. Heute meist Teil eines Botnetzes

- *Email Based*
    - Social engineering
    - Einfluss auf SMTP Infrastruktur
    - Eingeschränkt durch den Menschen

- *Exploit Based*
    - Kein Menschliche Interaktion erforderlich
    - Hohe Geschwindigkeit
    - TCP Verbindung oder UDP

Suspectible -> **I**nfected -> **R**ecovered

Trojan Horses

- Meist nach außen ein harmloses Programm
- TYP1 Meist versteckt in sinnvoller Software, aber mit Hintertür
- TYP2 Stehen alleine und locken durch interessante Informationen
- Unterschiedliche absichten
    - Spionieren
    - Rootkits
        - Versteckt Anwesenheit von Angreifer
        - Erlaubt es dem Angreifer später wieder zu kommen
        - Sammelt Informationen über die Umgebung
        - System Logging
        - System Monitoring
        - User Authentication
    - Kernal Rootkits
        - Verändert den Kernel
    - Remote Access
    - Zerstören von Datem

Spyware

- Software, welche Informationen über den Nutzer sammelt und an den Angreifer sendet
- Das Ziel ist, sensible Daten des Nutzer zu erhalten
    - Das Nutzer Verhalten muss dargestellt werden
    - Kann in Software versteckt werden
- Kann eine Firma sein, welche Informationen weiter verkauft

Bots und Botnets

- Eigenständige Programme
- Eingesetzt für IRC Channels
- Heutzutage werden Bots benutzt um ein botnetz aufzubauen für Ddos oder Spamming

Botnet Propagation

- Network worm
- Email attachment
- Social engineering
- Trjan version of program
- Drive-by download
- Existing backdoor
- Infected USB stick

Botnet Architektur

- Zentriert C&C
    - IRC
    - Webserver
    - Multiple Controlers
- Peer-to-Peer
    - Jeder Host kann Nutzer oder Proxy sein
    - Mehrstufige Hierarchie möglich

Botnet Defense

Horizontale Technik: zwei oder mehrere Host haben den selben Netzwerktraffic

Vertikale Technik: Analyse des Kommando- und Kontrolltraffics
Signature-based (AV)
Rule-based: Ports werden geblockt und kontrolliert
Netzwerkinhalt: Ddos Befehle werden gesucht
Netzwerktrafficanalyse: IP-Blacklist, DNS Anfragen
Angriffskommandos: abschalten einer Kommando-Infrastruktur
Honeypots: Erlaubt Opfern den Angriff zu analysieren

Malware Detection

- AV Scanner
    - Große Liste von Malware-Signaturen
    - Signaturen sind ASCII Strings oder Code Segmente

- Es gibt keine Möglichkeit jeden Virus der jemals existieren wird zu erkennen (Cohens Law)
- Klassifizierung
    1. Signatur Scanner
    2. Heuristik
    3. Identifizierungs Aktion
    4. Kombination/Hyprid Annäherung
- Signatur durch Keywordtree nach Aho-Corasick Algo.
- Untersuchen des Verhaltens

10. Honeypots

Dienstag, 10. Februar 2015 13:51

Grundidee

- Honeypots sind keine Produktivsysteme
- Ein Honeypot sollte keine Handlungen im Netzwerk machen
- Wenn nun ein Honeypot eine Interaktion im Netzwerk betreibt, ist dieser meist befallen

Ziele

- Aufzeigen der Handlung eines Angreifers während eines Angriffes
- Neue Angriffe erkennen
- Filtern von Informationen um den Angriff zu erkennen
- Sammeln von Malware
- Einen Angreifer vom Angreifen abhalten

Pros und Cons

Man erhält genaue Informationen über den Angriff	Es werden nur abgefangene Interaktionen verwertet
Man fängt nur böse Interaktionen ab	Gefahr vom Angreifer übernommen zu werden
Ermöglicht im Stillen zu handeln	

Honeypot

Honeyd

- Virtuelle Honeypots auf einem Host in freien IP Adressen

Nepethes

- Simuliert angreifbare Netzwerkservice
- Es werden nur erforderliche Teile dieser Service simuliert

Recap of Exporation

1. Angreifer überlastet den Service um anfällig zu werden
2. Angreifer sendet packet welches der Service ausführt
3. Das Payload lädt meist andere Software auf den Opferservice nach

Shellcode Modules

1. Payload wird analysiert
2. URL werden ausgeführt
3. Shellcode wird ausgeführt

Download Modules

1. Download der Malware über URL
2. Verschiedene Module für verschiedene Downloadmethoden

Submission Modules

1. Verwenden der Datei
2. Auf Festplatte schreiben
3. In Datenbank aufnehmen

Honeyclients

Simulieren Prozesse welche Nutzerseitig laufen und erzeugen eine Interaktion um Angegriffen zu werden
Senden Anfragen an verdächtige Server und Analysieren Response

Honeynets

- Eine Architektur von Honeypots
- Hohe Interaktion von Honeypots soll detaillierte Informationen sammeln

- Kontrollierte reale Computer
- Für den Angriff entwickelt
- Echte Anwendungen und Service werden ausgeführt

Herausforderungen

1. **Data Control**
   Angreifer in das Honeynet lassen und Traffic kontrollieren
2. **Data Capture**
   Alles aufzeichnen ohne erkannt zu werden
3. **Data Collection**
   Daten von mehreren Honeynets zusammen tragen
4. **Data Analysis**
   Daten in nutzbare Informationen umwandeln

Das Honeynet Projekt

Ziele

Erkenntnis Aufzeigen welches Risiko im Netz herrscht
Informationen Lehren und informieren über Gefahren
Forschung Organisationen die Möglichkeit geben zu lernen

Organisation

- Kein Profit
- Weltweite Mitarbeit

11. Angriffe Teil1

Donnerstag, 12. Februar 2015 12:24

TOCTOU (Race Condition)
LD-PRELOAD
BUFFER OVERFLOW — Funktion bei der ein BOF leichter ist
↳ Ret2Libc — strcpy, strcat, sprintf, char*, fread
↳ Format String

TOCTOU

TimeOfCheck < TimeOfAttack < TimeOfUse
Rechte überprüft → Datei link ändern → falsche Datei wird geöffnet
Im Quelltext nach sleep() suchen
Befehle touch dummy
ln -s dummy ptr } Vorbereitung
rm ptr
ln -s etc/passwd ptr } Angriffsscript

LD-PRELOAD

Programme benutzen Funktionen die in SharedLibraries (SL)
LD-PRELOAD ermöglicht Funktionen vor der SL laden

Nun kann man neue Funktion oder alte überschreiben in die LDP laden und so dafür sorgen, dass bekannte/benutzte Funktion einen anderen Ablauf haben

Neue Datei mit neuer Funktion
↳ compile
gcc -shared -FPIC -o datei.so datei.c
↳ include in LDP

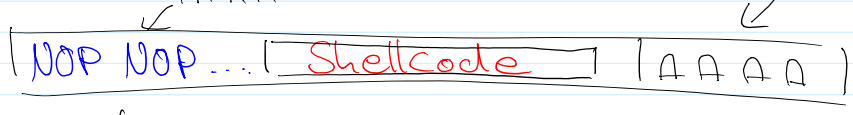
LD_PRELOAD=./datei.so /bin/ls ← wir denken /bin/ls, weil wir ls im Beispiel ersetzt haben

BUFFER OVERFLOW

Shellcode in den Speicher schreiben
Sicherheitslücke finden um Rücksprungaddresse ändern
Nun für Dumme:

Ein Programm hat die Möglichkeit eine Eingabe in den Stack zu schreiben
Wird nun die Eingabe großer als der reservierte Speicher gewählt
Schreibt man nun so viel Inhalt, dass Variblen oder sogar die Rücksprungadresse überschreibt, kann man den Programmablauf verändern

Wie funktioniert das jetzt genau



Wir haben es geschafft unsere Eingabe inklusive Shellcode in den Stack zu laden nun müssen wir die Rücksprungadresse so ändern, dass diese auf ein NOP vor unseren Shellcode zeigt. Wird nun über die RSAdresse zu einer NOP gesprungen, werden alle folgende NOPs abgearbeitet (nicht passiert) bis unser Shellcode ausgeführt wird
Die Rücksprungadresse ist in Little-Endian-Format:
0xAB 0xCD 0xEF 0xGH $\xrightarrow{\text{LEF}}$ 0xGH 0xEF 0xCD 0xAB

Gegenmaßnahmen
siehe Seite 7 (Buffer Overflow)

Bei NX benutzen wir Return 2 Libc
Da kein Shellcode aus dem Stack ausgeführt wird
Wir helfen uns mit der Funktion system
system kann die gegebenen Parameter als Funktion ausführen
d.h. wir laden via BOF unsere Funktion als Parameter auf den Stack, es folgen Platzhalter und an der Position der Rücksprungadresse

Format Strings

printf("%s", i) i = AAAA
Ausgabe AAAA

printf("AAAA%n", &i)
Anzahl 4
Speichert in die Variable auf die i zeigt die Anzahl der Zeichen die vor %n liegen