Informationssicherheit: Schwachstellen, Bedrohungen, Angriffe

Sicherheitsprobleme

- Zugriffskontrolle setzt Sicherheitsziele (wie z.B. Integrität, Vertraulichkeit) durch
- Nun: Umgehung der Mechanismen zur Zugriffskontrolle
- Typische Sicherheitsprobleme:
  - Buffer-Overflows (Unchecked buffer)
  - SQL-Injection
  - XSS-Probleme
  - TOCTOU-Angriffe
  - Bösartige Programme (Viren, Würmer, Trojanische Pferde)
  - Spoofing
  - DoS-Angriffe

Verwundbarkeit und „Exploit“

- Nicht jede Verwundbarkeit ist einfach zu nutzen
- „Exploit“: Programm/Verfahren zur Nutzung der Verwundbarkeit

Heiß diskutierte Frage:

- Soll man Sicherheitsprobleme veröffentlichen?
  - Ja, weil sie sonst nicht in Ordnung gebracht werden?
  - Nein, weil sie vermutlich sofort ausgenutzt werden?
- Soll man den „Exploit“ gleich mitliefern?

CERTs und Advisories

- CERT = Computer Emergency Response Team
  - Behandelt Sicherheitsvorfälle, IT-Sicherheitsfeuerwehr
  - Typische CERT-Betreiber: Große Unternehmen, Behörden, Staaten
- Advisory: Recherchierte Mitteilung über Sicherheitsproblem und mögliche Abhilfen (z.B. zu finden unter: https://www.kb.cert.org/vuls)
  - Wird meist erst veröffentlicht, nachdem Hersteller Abhilfen (Patches) zur Verfügung gestellt haben
  - Etwas schwerfälliger Vorgang
  - Bleibt meist nebulös, worin genau das Problem besteht
- Problem: Patches lassen sich analysieren
  - Exploits stehen ca. zehn Stunden nach Advisory zur Verfügung
  - „Patch race“

Software im Netz

- Server sitzen da und warten auf Verbindungen
  - Meist identifiziert über IP-Adresse und Port-Nummer
- Client stellt Verbindung her, stellt Anfrage

- „Remote exploit“: Anfrage an Server, die unmittelbar zum unkontrollierten Zugriff führt
- „Local exploit“: Angriff wird erst nach Authentisierung und Erlangung eigener Zugriffsrechte (Programmausführung) möglich

Buffer-Overflows

- Häufigste Einbruchmethode in Server (insbesondere in Web-Server)
- Altbekanntes Problem (schon in den 60er Jahren bekannt)
- „Attack of the decade“ (Bill Gates)
- Die meisten Viren/Würmer nutzten Buffer-Overflows aus (Morris-Wurm, Code Red, Blaster)
- Ziel: Einschleusen von Code
- Ausnutzen von Programmierfehlern

Nicht korrekt geprüfte Eingaben

```
char buf[42];
gets(buf);
```

- **Problem**:
  - C-Funktion `gets` überprüft nicht die die Länge der Eingabe („unchecked buffer“)
  - Ist die Länge der Eingabe größer als 41 (Nullterminierung beachten!), dann wird umgebender Speicher überschrieben.
  - Lokale Variablen sind auf dem Call-Stack
- Durch Verwendung von `fgets(buf,42,stdin)` wird das Problem vermieden

Ein lokaler „unchecked buffer“ (historisches Beispiel)

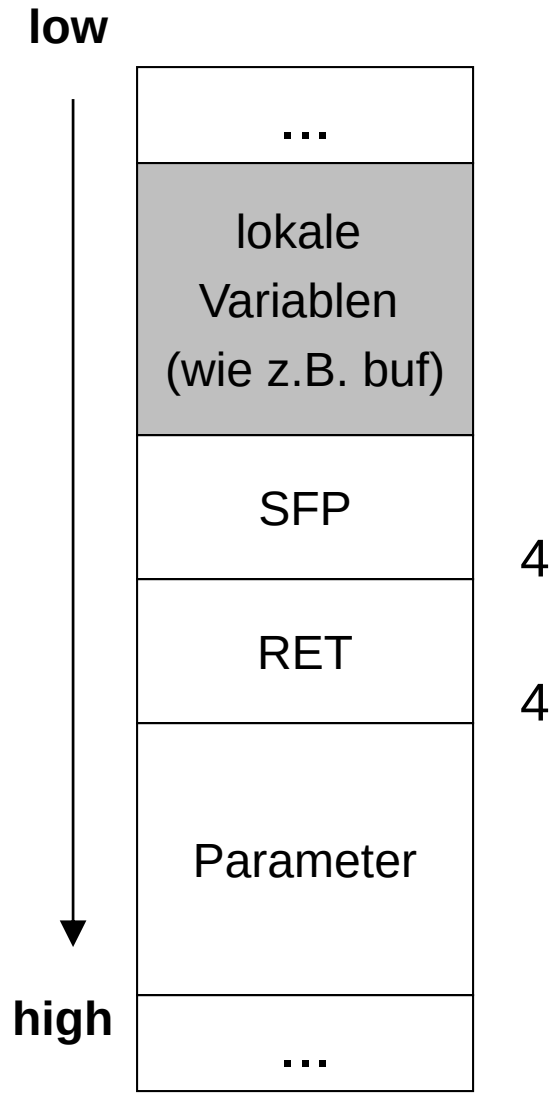
- UNIX Version 6 (ca. 1975), Login-Programm:

```
char user[100], passwd[100], correct[100];
gets(user); getpwnam(…); strcpy(correct, …);
gets(passwd);
if (strcmp(crypt(passwd, …),correct)) …
```

- Eingabe eines 108 Zeichen langen Passworts schreibt über das Ende von `passwd` hinaus und überschreibt den Vergleichswert `correct` aus dem Passwort-File
- Speziell fabriziertes Passwort, das in seine eigenen letzten 8 Zeichen verschlüsselt, ist Generalschlüssel!

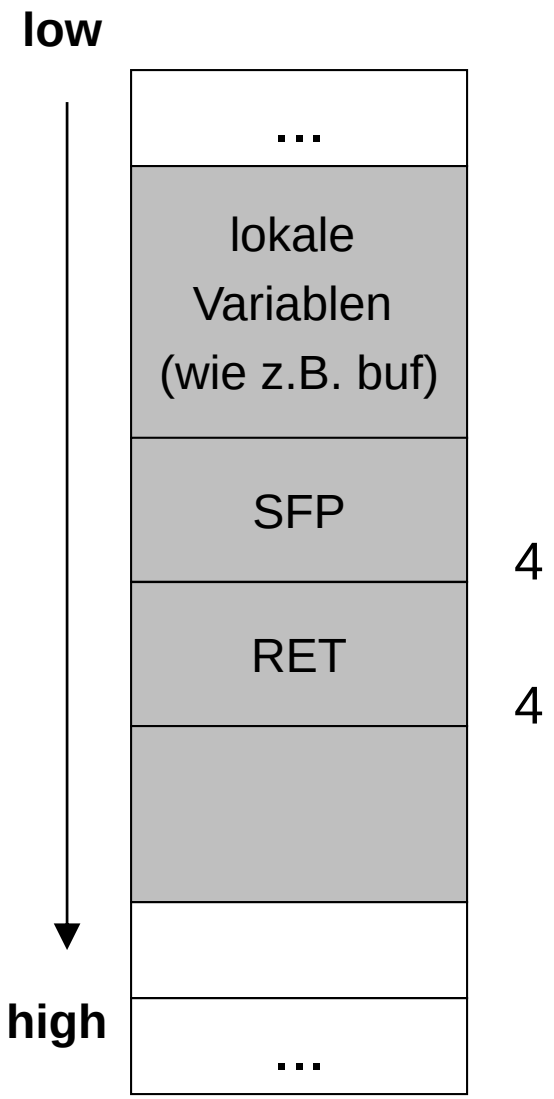
Call-Stack

- Lokale Variablen von Unterprogrammen werden auf Call-Stack abgelegt
- **Beispiel**: Stack für einen Unterprogrammaufruf bei der x86-Architektur
- RET: Rücksprungadresse
- SFP: Stack Frame Pointer



Überschreiben der Rücksprungadresse

- RET kann durch entsprechend präparierten Eingabestring so manipuliert werden, dass Rücksprung in den Exploit-Code führt
- Exploit-Code wird in Stack untergebracht
- Beispiel für Exploit-Code unter Linux/Unix: `execl` zur Ausführung einer Shell auf angegriffenem System
- Nützlich, wenn das Programm mit der Buffer-Overflow-Verwundbarkeit mit root-Rechten läuft (Daemon oder s-Bit für den Besitzer root gesetzt)



Präparation des Eingabe-Strings

String-Anfang **String-Ende**

Füllzeichen	**Neue Rück-sprungadresse (u.U. mehrfach)**	**NOPs**	**Code des Angreifers (z.B. Shell)**

Landing-Pad aus Nulloperationen (NOPs):

Landezone vor dem eigentlichen Angriffscode, weil die *genaue* Adresse des Angriffscodes oft nicht vorhersagbar ist.

Fallstricke für einen Angreifer

- Platz auf Stack zu klein? Eher unproblematisch:
  - Maschinencode ist **kompakt**
  - **Viele Funktionen** stehen zur Verfügung,
    einfache Nutzung vorhandener Dienste:
    - z.B. Windows: viele Bibliotheken (DLLs) stehen bereits im Speicher und sind an den befallenen Prozess gebunden
    - Nutzung aller API-Funktionen, die bereits zum Programm gebunden sind
      z.B. *LoadLibrary*: **Nachladen** beliebiger Funktionen möglich!
- Was ist, wenn die Rücksprungadresse oder der Exploit-Code spezielle Zeichen wie `'\0'` oder `'\n'` enthält?
  - Viele Möglichkeiten, die gleiche Funktion zu kodieren

Gegenmaßnahme: NX-Bit

- Idee: Ausführen von Code auf dem Stack verbieten
  - Erweiterungen in aktuellen Prozessoren (AMD64: NX, Intel x86: Execute Disabled/ XD)
  - Windows: Data Execution Prevention (DEP)
  - Achtung: Datenbereiche z.B. für Shared Libraries oder Code-Generierung (JIT-Compiler!)
  - Einzelne Programme benutzen sogar Stack für Code
- Gegen-Idee: „Rücksprung“ in Bibliothek, geeignete Parameter können den gleichen Effekt haben wie Code auf dem Stack
  - Aber erhebliche Erschwerung komplexer Angriffe

Gegenmaßnahme: Canaries

- Funktionsprolog legt einen zufälligen Wert (Canary) unmittelbar neben die Rücksprungadresse
- Vor Rücksprung wird überprüft, ob Canary nicht verändert worden ist
  - Wenn nicht: Log-Meldung/Abbruch
- Windows: Stack Cookies
- Nachteil: Kosten für Prozeduraufruf steigen

Gegenmaßnahme: ASLR

- **Problem:** Angreifer springt in Bibliothek, beispielsweise Aufruf von `system()` in Unix/Linux

- **Lösungsidee:**
  Address Space Layout Randomization (ASLR)
- Z.B. Shared Libraries (Bibliotheken), Stack an zufälligen Adressen positionieren
- Angreifer kann schlecht die Position von Systemcalls erraten
- Unterstützung in modernen OS wie BSD Unix, Linux, Windows, Android, iOS
- Reaktive Maßnahme – keine Prävention des eigentlichen Problems

Fazit: Buffer-Overflows

- **Haupt-Ursache für Buffer-Overflows**: Programmierfehler:
  Verwendung von vordefinierten Bibliotheksroutinen in Programmiersprachen wie C oder C++ ohne Bereichsprüfung:
  - strcpy(), strcat(), gets() in C
  - Es gibt Alternativen strncpy(), strncat(), fgets().

- **Bemerkung**:
  Die meisten BS-Dienste (Unix, Windows) sind in C, C++ programmiert.
  (z.B. Windows XP: ca. 40 Millionen Lines of C-Code —
  Richtwert: 5-50 Bugs pro 1000 Lines of Code!)
- Fast alle Würmer nutzen Buffer-Overflows aus

Ausblick: Buffer-Overflows

- So langsam haben sich SW-Hersteller besser auf die Problematik eingestellt
  - Werkzeuge zur statischen Software-Analyse (Fortify SCA, Coverity Prevent, Checkmarx, Veracode, Clang Static Analyzer…)
  - Was ist mit Embedded Systems und IoT-Geräten?
- Nun geht es für einen Angreifer an die etwas höher hängenden Früchte wie z.B.:
  - Heap-Overflows, Integer-Overflows
  - Race Conditions (→ später in dieser Vorlesung)
  - Architekturelle Lücken: Mangelnde / inkonsistente Zugriffskontrolle in Multi-Tier-Applikationen (z.B. SOA), fehlerhafte Umsetzung von Sicherheitsprotokollen wie z.B. WPA2-Lücke (Krack; Okt. 2017)

SQL-Injection

- Viele Web-Anwendungen (z.B. PHP) speichern Kundendaten, Login-Informationen in Datenbanken
- Die Daten werden von den Benutzern in Eingabefelder eingegeben wie z.B. E-Mail-Adressen
- PHP bietet z.B. eine bequem benutzbare Datenbankschnittstelle, um die eingegebenen Daten direkt in die Datenbank auf dem Web-Server zu schreiben
- Wie bei den Buffer-Overflows: Angreifer definiert Daten!

... Angreifer gibt die Daten ein

Beispiel:

Eingabefeld für eine E-Mail-Adresse in einem Web-Formular für Kunden der Firma „Unsicher GmbH“

Angenommen, PHP-Anwendung nutzt SQL-Statement:

```
SELECT email, passwd, login_id, name
FROM members
WHERE email='Daten vom Netz'
```

unsere Eingabe = `bla@tzi.de'; DROP TABLE members;--`

Im SQL-Statement:

```
...WHERE email ='bla@tzi.de'; DROP TABLE members;--'
```

Fazit

- Ursache: nicht validierte Eingaben werden als Programmcode ausgeführt
  - Variante: PHP- oder Perl-Code
- SQL-Injection-Angriffe sind nicht einfach durchzuführen:
  - Woher kennt der Angreifer die Namen der DB-Relationen und Attribute? (PHP-Fehlermeldungen helfen allerdings)
  - Oft nur Anrichten von Schaden ($\rightarrow$ Datenintegrität)
- Trotzdem: Fast täglich Meldungen über neue SQL-Injection-Angriffe in Mailinglisten für Sicherheitsprobleme wie z.B. Bugtraq

Cross Site Scripting (XSS)

- Webseiten können Skripte enthalten
  - Werden auf Kunden-Browser ausgeführt
  - Haben Zugriff auf Cookies der Website
- Webseiten können benutzerdefinierte Daten enthalten
  - Vorherige Eingaben eines anderen Benutzers
  - Evtl. auch unerwartete Reaktion auf URL-Parameter
- Angriff: Angreifer unterschiebt Opfer ein Skript

Durchführung eines Angriffs

- **Beispiel**:

  http://auction.example.com/filename.html liefert eine Webseite zurück mit der Meldung

  `404 page does not exist: filename.html.`
- Angreifer schickt dem Opfer nun einen präparierten Link (z.B. per E-Mail):

  http://auction.example.com/<script>alert('hello')</script>
- Beim Abruf des Links wird das Skript zurückgeliefert und aufgerufen (und zwar nicht auf dem Webserver, sondern auf dem Rechner des Opfers)

Beispiel für verwundbaren JSP-Server-Code

```
<c:if test="${param.sayHello}">
     <!—Let's welcome the user ${param.name} -->
     Hello ${param.name}!
</c:if>
```

param.name wird einfach ungeprüft vom Server zurück zum Client gespielt (Beispiel aus Buch von Chess und West)

Fazit: XSS-Angriffe

- Angreifer kann auf Opferrechner beliebige Javascript-Befehle ausführen:
  - Cookie auslesen und woanders ablegen
  - Eingabefenster für Passwörter simulieren
- Session-Übernahme möglich durch Stehlen eines Session-Cookies (im Auktionsbeispiel evtl. besonders kritisch)
- Angegriffener: primär der Nutzer, aber z.B. durch Rufschädigung auch der Betreiber der Website
- Betreiber der Website sollte solche Fehler beseitigen
- Schutz: Validation von Eingaben (durch den Betreiber der Web-Site)

Zwischenfazit

- Ursache für Buffer-Overflows, SQL-Injection-Angriffe sowie XSS-Angriffe: ungeprüfte Eingaben

- ALSO:
  Überprüfen von Eingaben, Überprüfen von Eingaben, Überprüfen von Eingaben!!!
  - Nicht nach Problemen suchen (der Angreifer hat mehr Phantasie), sondern nur gesunde Eingaben durchlassen
    (vgl. UTF-8-Angriffe)

TOCTOU-Angriffe (1)

- TOCTOU: Time of Check, Time of Use
- Anderer, allgemeinerer Begriff: Race Conditions (Wettlaufsituationen)
- Basiert auf Nebenläufigkeit in Betriebssystemen und Anwendungen
- Operationen oft nicht atomisch

- **Beispiel**: `mkdir` in alten Unix-Systemen
  - 1. Schritt: Anlegen des inode für das Verzeichnis
  - 2. Schritt: Festlegen des Besitzers (des Verzeichnisses)

- Angreifer kann nach erstem Schritt in einem zweiten Prozess einen Link auf `/etc/passwd` setzen,
- In Schritt 2 wird dann der Angreifer zum Besitzer von `/etc/passwd`

TOCTOU-Angriffe (2)

- TOCTOU-Angriffe: Es kommt für den Angreifer auf den richtigen Zeitpunkt an
  - Ausprobieren, Angriff evtl. oft wiederholen, Angriff automatisieren
- Häufiges Muster für Verwundbarkeit:
  1. Überprüfung der Zugriffsrechte (Time of Check)
  2. Durchführung der sicherheitskritischen Operation (Time of Use)
- Zugriffskontrolle kann ausgehebelt werden

Spoofing

- Statt Berechtigung zu erschleichen:
  Vorgeben, ein Berechtigter zu sein
- Manche Systeme prüfen nur die Absender-IP-Adresse
  - Bei UDP extrem leicht zu fälschen
  - Bei TCP schwieriger, aber in bestimmten Fällen möglich
- Session hijacking: Verbindung nach der Authentisierung übernehmen
  - Einfache Gegenmaßnahmen in TCP, wenn kein Abhören möglich
  - Echter Schutz nur kryptographisch möglich

(D)DoS-Angriffe

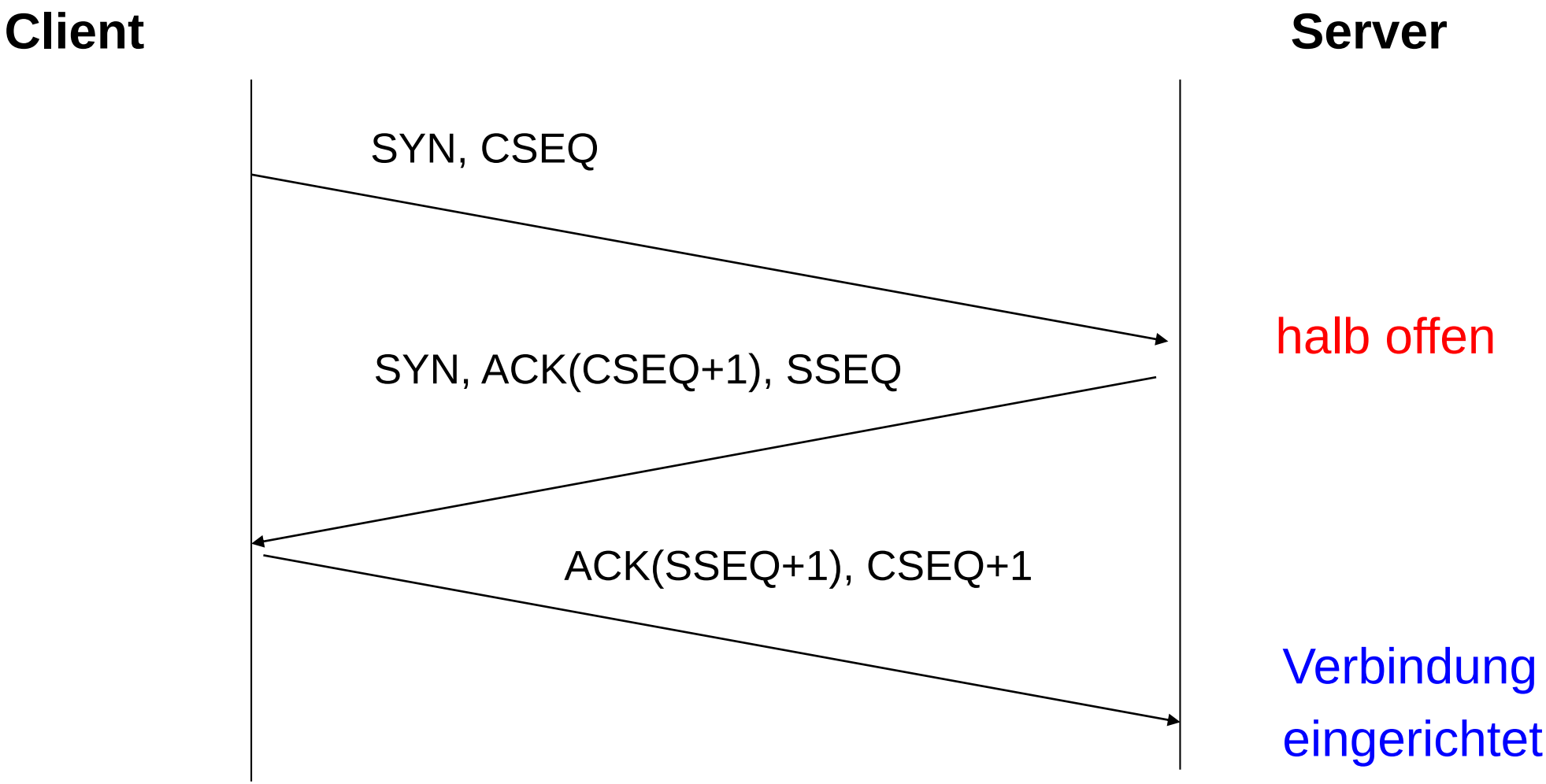
- Denial-of-Service: Angriff auf Sicherheitsziel

  Verfügbarkeit
- Server zum Absturz bringen
  - Programmfehler wie unchecked buffer nutzen
- Sicherheitsmaßnahmen aktivieren
  - Z.B. Account-Sperre nach dreimaliger Fehleingabe des Passwortes
- Server (oder Netz) überlasten
  - DDoS: Distributed DoS: Farm übernommener „Zombies“

Syn-Flood-Angriff

- Ziel: Server verstopfen, ohne Identität der angreifenden Maschinen preiszugeben
- TCP: Schutz durch three-way-handshake (→ nächste Folie)
  - Verbindung kommt nur zustande, wenn Antwort vom Client stimmt
  - Absenderadresse kann nicht leicht gefälscht werden
- Idee: Nur SYN-Paket absenden
  - Absenderadresse leicht zu fälschen
  - Server muss Zustand halten (Timeout nach Minuten)
- 1 Gbit/s ≈ 3E6 Pakete/s ≈ 0.5E9 halboffene Verbindungen

TCP-Handshake



Trojanische Pferde

- Analog zur griechischen Sage:
  - Ein nach außen hin nützlich erscheinendes Programm, das eine verborgene Schadfunktionalität enthält
- Selbständiges Programm (im Gegensatz zum Virus)
- „Werkzeuge zur Fernwartung“ (remote administration)
- **Beispiele**:
  - BackOrifice (Windows95/98, August 1998, Cult of the Dead Cow), BO2K (Windows, Juli 1999)
  - SubSeven

    Funktionalität: Manipulieren der Windows Registry, entferntes Lesen und Schreiben von Dateien, Keylogger, Port Scans vom Opferrechner aus, ...
- Ständige Veröffentlichung neuer Trojanischer Pferde auf Cracker-Webseiten

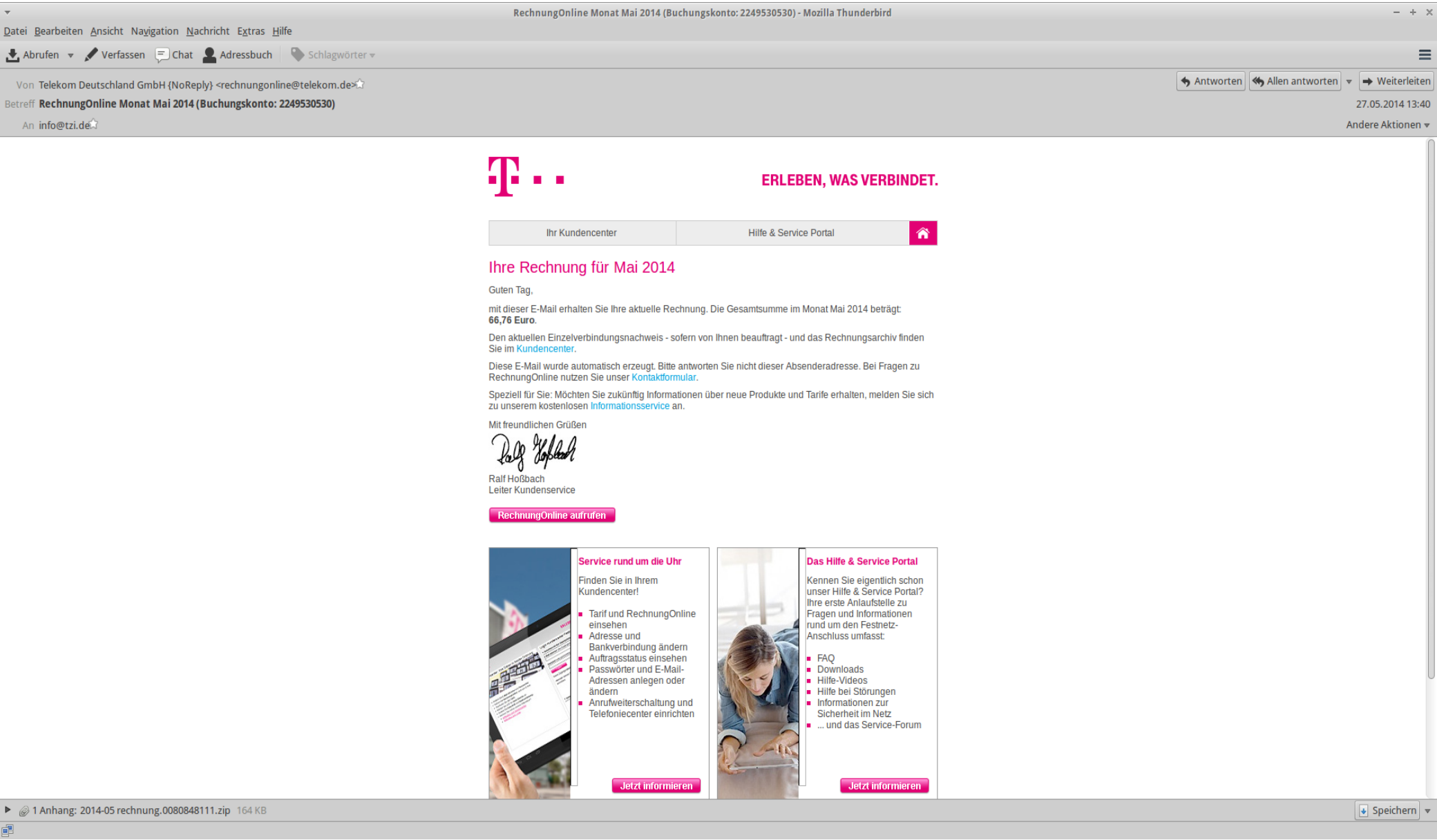
Weitere Varianten von Trojanischen Pferden/Malware

- Spyware
  - Ziel: **Verdecktes Ausspähen** des Nutzers
  - Oft in Form eines Trojanischen Pferdes
  - Beispiele: „Web-of-Trust“-Add-on für Browser, Mitschneiden von Browser-Verläufen; Apps mit unautorisiertem Mikrofon- und Kamerazugriff für Smartphones

- Ransomware
  - Englisch: „ransom“ – Lösegeld
  - Erpressersoftware: z.B. Entschlüsseln von Daten gegen Lösegeld

Social Engineering

- Angreifer überzeugt das Opfer, etwas zu tun, was er will
- Beispiele:
  - Erfragen von Patientendaten per Telefon
  - Phishing
  - gefälschte E-Mail-Adressen (E-Mail kommt vermeintlich von einem Bekannten, vgl. Verbreitung von Würmern)
- **I Love You** nutzte mehrfach Social Engineering-Techniken, z.B.
  - `LOVE-LETTER-FOR-YOU.TXT.vbs` (Windows lässt oft bekannte Endungen weg, so dass das VBS-Script wie eine Text-Datei aussah)
- Kevin Mitnick: „I was so successful in that line of attack that I rarely had to resort to a technical attack“
- Schutz: Schulung von Mitarbeitern, Sensibilisierung


RechnungOnline Monat Mai 2014 (Buchungskonto: 2249530530) - Mozilla Thunderbird
Datei Bearbeiten Ansicht Navigation Nachricht Extras Hilfe
Abrufen Verfassen Chat Adressbuch Schlagwörter
Von Telekom Deutschland GmbH {NoReply} <rechnungonline@telekom.de>
Betreff RechnungOnline Monat Mai 2014 (Buchungskonto: 2249530530)
An info@tzi.de
Antworten Allen antworten Weiterleiten
27.05.2014 13:40
Andere Aktionen
ERLEBEN, WAS VERBINDET.
Ihr Kundencenter
Hilfe & Service Portal
Ihre Rechnung für Mai 2014
Guten Tag,
mit dieser E-Mail erhalten Sie Ihre aktuelle Rechnung. Die Gesamtsumme im Monat Mai 2014 beträgt: 66,76 Euro.
Den aktuellen Einzelverbindungsnachweis - sofern von Ihnen beauftragt - und das Rechnungsarchiv finden Sie im Kundencenter.
Diese E-Mail wurde automatisch erzeugt. Bitte antworten Sie nicht dieser Absenderadresse. Bei Fragen zu RechnungOnline nutzen Sie unser Kontaktformular.
Speziell für Sie: Möchten Sie zukünftig Informationen über neue Produkte und Tarife erhalten, melden Sie sich zu unserem kostenlosen Informationsservice an.
Mit freundlichen Grüßen
Ralf Hoßbach
Leiter Kundenservice
RechnungOnline aufrufen
Service rund um die Uhr
Finden Sie in Ihrem Kundencenter!
Tarif und RechnungOnline einsehen
Adresse und Bankverbindung ändern
Auftragsstatus einsehen
Passwörter und E-Mail-Adressen anlegen oder ändern
Anrufweiterschaltung und Telefoniecenter einrichten
Jetzt informieren
Das Hilfe & Service Portal
Kennen Sie eigentlich schon unser Hilfe & Service Portal? Ihre erste Anlaufstelle zu Fragen und Informationen rund um den Festnetz-Anschluss umfasst:
FAQ
Downloads
Hilfe-Videos
Hilfe bei Störungen
Informationen zur Sicherheit im Netz
... und das Service-Forum
Jetzt informieren
1 Anhang: 2014-05 rechnung.0080848111.zip 164 KB
Speichern

Viren (1)

- In der Biologie ist ein Virus
  - Ein Mikro-Organismus, der auf eine lebende Wirtszelle angewiesen ist,
  - Keinen eigenen Stoffwechsel besitzt und
  - Fähig ist, sich zu reproduzieren
- Eigenschaften sind direkt auf Computerviren übertragbar:
  Computervirus: von F. Cohen 1984 eingeführter Begriff
- Historischer Begriff

Viren (2)

Computervirus:

- Nicht selbständiges Programm, d.h. es benötigt Wirt
- Beispiel für Wirt: meist ausführbare Dateien
- Besitzt Kopierfähigkeit, ggf. auch mutierend
- Enthält i.d.R. einen Schadensteil, d.h. Code zur Durchführung von Angriffen
- Kann Auslöser enthalten, d.h. Bedingung zur Aktivierung des Schadensteils (logische Bombe) wie z.B. Datum
- Enthält i.d.R. eine Kennung, z.B. Zeichenketten
- Virus-Code dient häufig zur gezielten Angriffs- (Einbruchs)-vorbereitung: u.a.
  - Infos sammeln, Ports öffnen, Shell-starten, ...

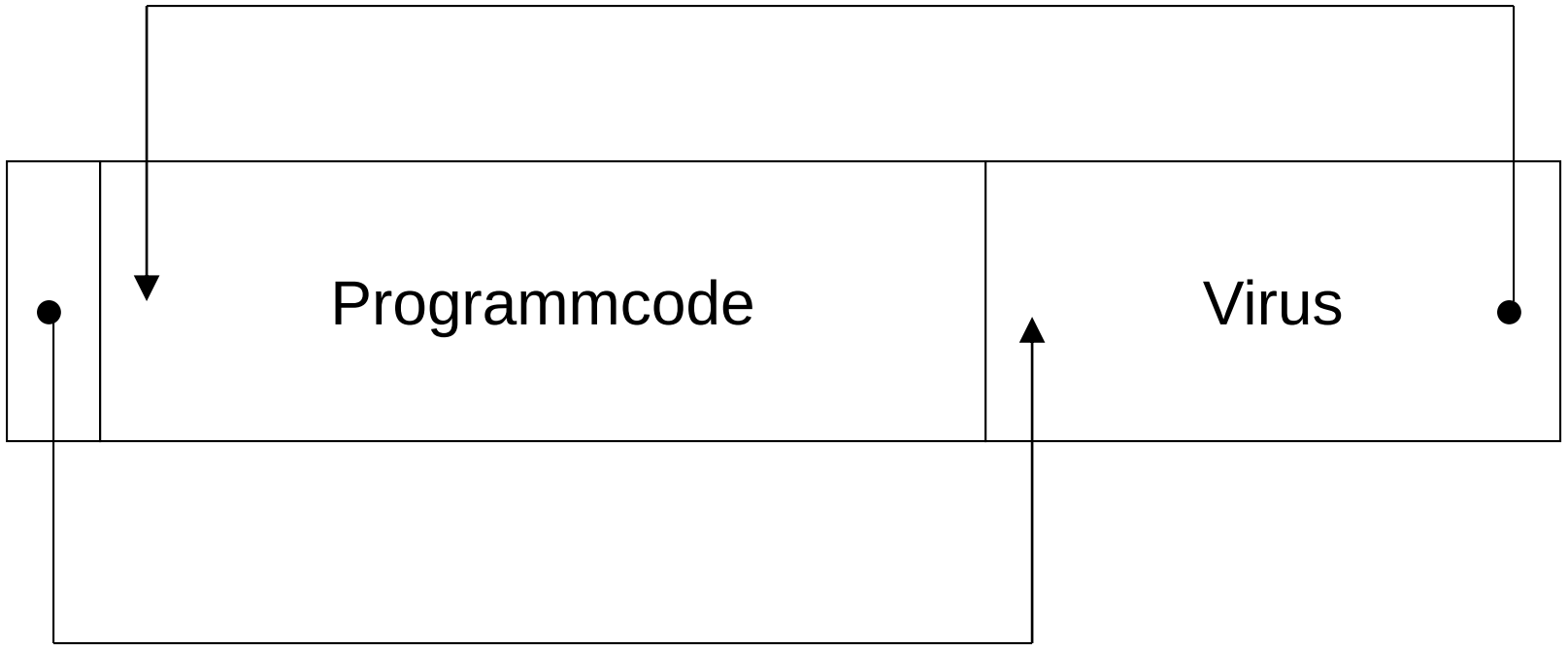
Virentypen

- Programmviren (Link-Viren): infizieren ausführbare Programme (z.B. .exe); Virus-Start mit dem Programm; s. folgende Folie
- Bootsektor-Viren: Virus wird resident geladen
- Makro- und Daten-Viren: u.a. bei MIME, .ps, .doc, .xls
  - Interpretative Ausführung von Code

    z.B. Starten eines Dateitransfers (von Festplatte) via FTP
  - Häufig: Verbreitung über das Netz, via E-Mail Attachments, Buffer-Overflow-Angriffe etc.
- Retro-Viren**:** gegen das Immunsystem (z.B. Viren-Scanner)

  mögliches Angriffsziel: Deaktivieren des Viren-Scanners
- Angriff anderer Systeme: Smartphone-Viren?
- Viren benutzen oft Spam, um sich schneller zu verbreiten (Würmer?)

Funktionsweise: Programmvirus



Gegenmaßnahme: Virenscanner

- „Signatur“ eines Virus erkennen
- Dateien im Eingang und Ausgang (z.B. Mail) und vor der Ausführung prüfen
- Aufwendig
- Tiefer Eingriff ins Betriebssystem
- Aktualität der Signaturen ist entscheidend
- Polymorphe Viren erfordern „probeweises Ausführen“
- Virenscanner sind komplex und haben eigene Verwundbarkeiten

Würmer

- Selbständig ablauffähiges Programm,
- Nutzt die Infrastruktur eines Netzes, um sich selbsttätig zu verbreiten
- Ausgangspunkte für Wurmangriff häufig: Buffer-Overflow-Angriff auf Systemprozesse, die ständig rechenbereit sind oder in regelmäßigen Abständen aktiviert werden
- **Beispiele**:

  1988: Morris-Wurm (befiel Unix-Rechner): Buffer Overflow: fingerd

  2000: I Love You-Wurm

  2001: Code Red Wurm: Buffer Overflow in MS-IIS

  2003: W32/Lovsan/MS Blaster-Wurm

  2008: Conficker

Gegenmaßnahme: Firewall

- Nur erwünschte Kommunikation zulassen
  - Windows: zu viele Interna sind von außen zugreifbar; Firewall betriebsnotwendig
  - Problem: Wie erkennen, was erwünscht ist?
- IDS/IPS: Signaturen von Angriffen erkennen
  - Vgl. Virenscanner
  - Insbesondere: Erkennung von Angriffen, nur wenn Signatur vorhanden
  - Was ist aber mit unbekannten Angriffen?

Können wir gewinnen?

- Angreifer braucht nur **eine** Sicherheitslücke
- Verteidiger muss jedes Loch finden und stopfen

- Reale Systeme sind zu komplex, dass sie fehlerfrei sein könnten
- Eindringen **erschweren**, **erkennen**, **bekämpfen**

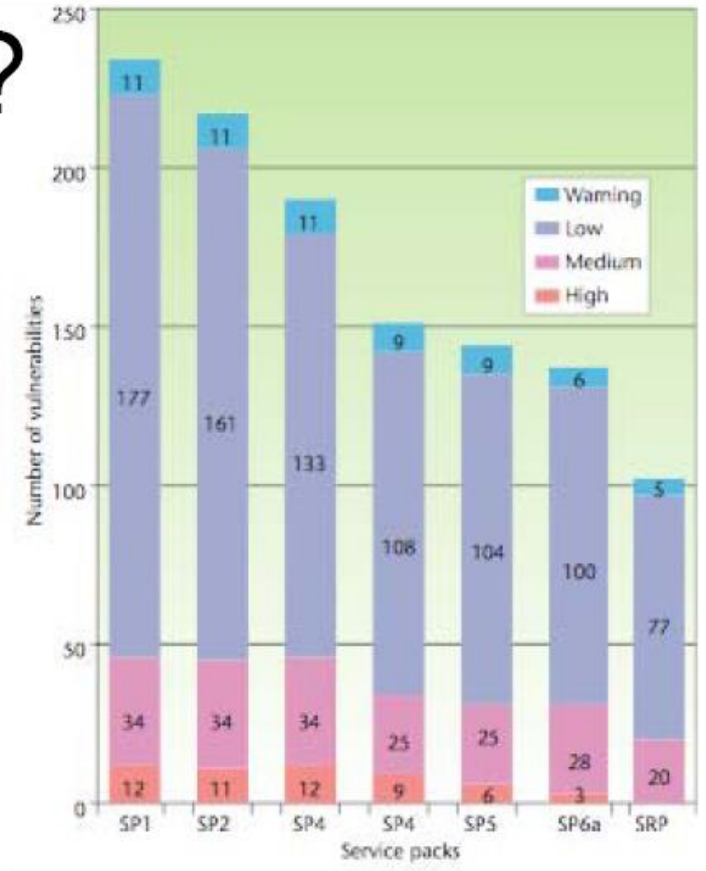


Figure 1. Security flaws. The number of security flaws in Windows NT 4.0 Service Packs 1 through 6a and the post-SP6a Security Rollup Package (SRP) according to risk levels.

Die vier Phasen eines Angriffs

- Aufklären
- Eindringen (exploit)
- Spuren verdecken (z.B. rootkit)
- Ausnutzen

- Gute Sicherheitstechnik versucht, alle diese Phasen zu **erschweren**, **erkennen**, **bekämpfen**

Interessante Links

- Computer Emergency Response Team (CERT) Coordination Center:

  https://www.sei.cmu.edu/about/divisions/cert/index.cfm
- SANS Institute: SANS TOP 25 Most Dangerous Software Errors

  http://www.sans.org/top25-software-errors/
- Zu Buffer-Overflows:

  http://insecure.org/stf/smashstack.html
- Zu Software-Sicherheitslücken im Allgemeinen:

  Brian Chess, Jacob West. 2007. *Secure Programming with Static Analysis*. Addison-Wesley Professional.