Software Security

Bauer Simon Gödde Falco Mösl Christian

Universität Salzburg
Fachbereich Computerwissenschaften

12. Juni 2017

Inhalte

Motivation

Ziel: Software-System soll auch bei Fehlern und Angriffen noch zuverlässig arbeiten

Bereiche:

- Kryptographie
- Hardware-Design
- Formale Bereiche
- Wirtschaft
- Psychologie
- Recht
- Software-Engineering

Anforderungen

Verfügbarkeit (availability): Stabilität, Verfügbarkeit von Funktionen, Daten bei Bedarf

Vertraulichkeit (confidentiality): Abschirmen geheimer Informationen gegenüber öffentlich zugänglichem Speicher

Integrität (integrity): Schutz des programmspezifischen Speicherinhalts und des Verhaltens vor öffentlichen Eingaben, keine Verfälschung von Daten

Autentizität (authenticity): Sicherstellen der Herkunft von Daten, Validierung der Echtheit

Ursachen & Folgen von Sicherheitslücken

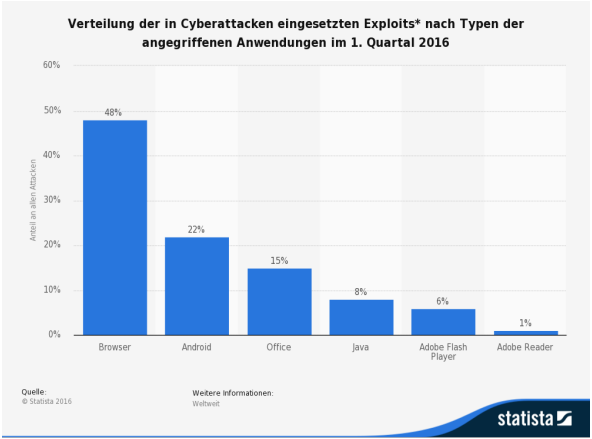
Ursachen

- Generierung: Compiler-Fehler (sehr unwahrscheinlich)
- Implementierung: Sourcecode-Fehler
  - Fehlende/Schlechte Kenntnisse
  - Zeitdruck
  - Mangelhaftes Testen

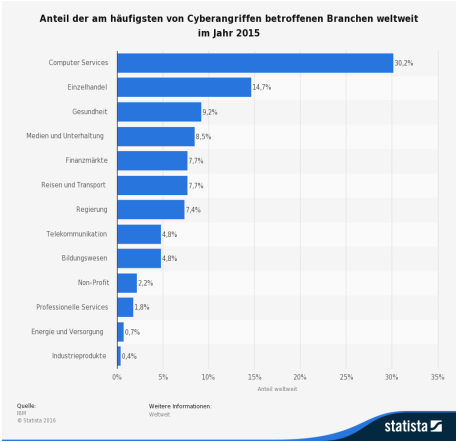
Folgen

- Falscher Output
- Absturz von Software/OS
- Beschädigung/Veränderung von Daten
- Datenraub
- Hardwarebeschädigung

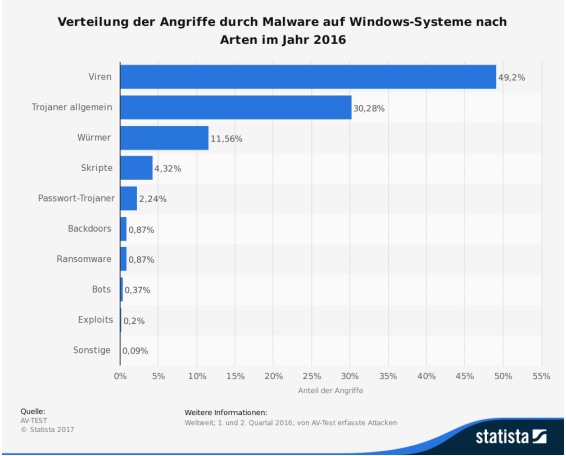
Statistiken



Statistiken



Statistiken



Exploits

Use sliders to select a range

01.08.1988 17.04.2017

Total Entries	37.751

Top 10 Platforms

php	18.391
windows	8.384
linux	2.475
multiple	2.179
asp	1.513
hardware	1.242
cgi	746
osx	309
lin_x86	307
unix	306

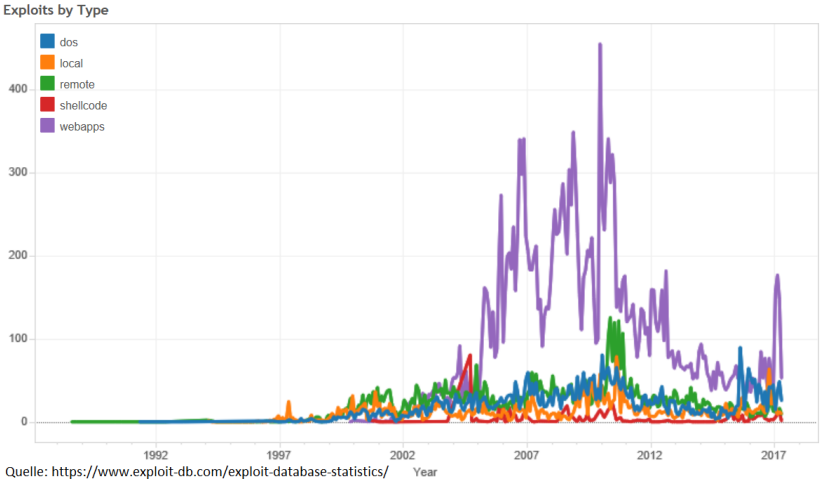
Top 10 Ports

Port	
80	1.270
21	148
8080	95
443	65
143	47

Exploits by Date

Quelle: https://www.exploit-db.com/exploit-database-statistics/

Exploits

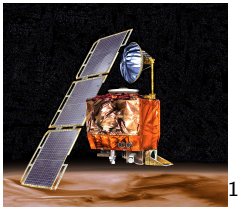


Quelle: https://www.exploit-db.com/exploit-database-statistics/

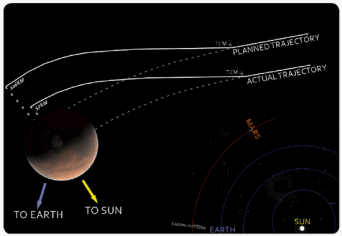
Geschichtliches

Mars Climate Orbiter

- 1998: unbemannte Marsexpedition
- Software Bug
- Falsche Umrechnung von englischen Einheiten ins metrische System
- Beschädigung der Kommunikationsinstrumente durch Atmosphäre
- Orbit um die Sonne oder verglüht
- 125 Mio USD verloren



1



2

[1] Quelle: Wikimedia.org
[2] Quelle: ni.com

Geschichtliches

Ariane 5 Flight 501

- Start einer Satelliten-Rakete
- Übernahme der Software des Vorgängers
- Fehlende Kompatibilität zu Ariane 5
- Versuch des Computers eine 64-Bit Zahl in 16-Bit Speicherzelle zu speichern $\Rightarrow$ Großer Overflow
- Folge: Primär- und Sekundärcomputer crashen
- Diagnoseinformationen werden als Flugdaten interpretiert $\Rightarrow$ Signal zur Schwenkung
- Komplette Zerstörung
- 290 Mio Euro Entwicklung + 500 Mio USD Satellit

Geschichtliches

The Patriot Missile Failure

- Golf Krieg: 1991
- Versagen einer Patriot Missile, eine feindliche Scud-Rakete abzufangen
- Umrechnung von Zehntel-Sekunden in Sekunden: Abschneiden der Zahl nach der 23sten Nachkommastelle
- Zeitdelta wird mit wachsender Uptime größer und ungenauer
- System erkennt die ankommende feindliche Rakete nicht als Ziel $\Rightarrow$ keine Gegenmaßnahmen
- 28 Tote und über 100 Verletzte

$\Rightarrow$ Software Security hat sehr hohe Priorität

Geschichtliches

Geld mit Exploits verdienen

- Apple: iOS-Bug-Bounty-Programm (200.000 USD) für ausgewählte Forscher
- Exodus Intelligence (500.000 USD)
- Regierungen, Behörden
- Google Chrome, Microsoft Edge, Firefox, Windows 10, Adobe Reader, Adobe Flash
- Zerodium: 1,5 Mio USD für iOS-Sicherheitslücke

Injection

Definition: Versuch eines Angreifers, Daten in laufendes Programm einzuschleusen.

Bereiche: SQL, OS, HTML, ...

Problem: Programme haben oft hohe Rechte

Hoher Prozentsatz an Sicherheitslücken bestehen aus Injection-Möglichkeiten.

Grundlagen

- Ungeprüfte Daten (User-Input, URL-Parameter, ...)
  ⇒ werden oft ungeprüft an Interpreter weitergegeben oder gespeichert
  (evtl. verändert, geparsed, etc... Schwachstelle schwer zu finden)
- Kontext (Abschnitt, der Input erwartet)
  ⇒ Angreifer will aus Kontext ausbrechen
- Parser (nimmt Input entgegen und gibt es an Interpreter weiter)
  ⇒ Verstehen des Parsers wichtig für Developer
  (aber auch für Angreifer)

Maßnahmen

Verifizierung: Prüfen, ob Input auch dem erwarteten Input entspricht
(je mehr Möglichkeiten man erwartet, desto komplizierter)

Escaping: kritische Symbole unschädlich machen
(z.B. “ $\Rightarrow$ \“ in Java)

Arten von Exploits

- SQL-Injection
- Buffer-Injection
- Command-Injection
- Cross-Site Scripting
- Cross-Site Request Forgery

SQL-Injection

Angreifbar: Frontend einer SQL-Datenbank

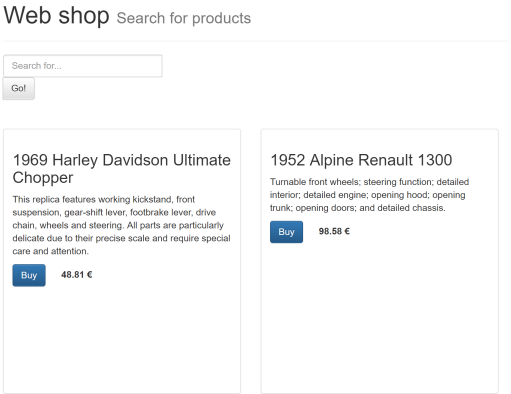
Ziel: Versuch, SQL-Befehle auszuführen

Exploit: Möglichkeit diese Befehle im Frontend einzufügen

Effekt:

- Zugriff auf Datenbank (lesend/schreibend)
- Steuerung des SQL-Systems

Demo - Webshop



Problem

Einlesen des User inputs:

```
<?php
$description = "%";
if ($_SERVER["REQUEST_METHOD"] == "GET" && isset($_GET["desc"])) {
    $description = "%" . $_GET["desc"] . "%";  // needs to be escaped
}

$sql = "SELECT productName, productDescription, buyPrice FROM "
     . "products WHERE productName LIKE '" . $description . "'";
?>
```

Maßnahmen

1. Feste Statements mit variablen Parametern
   Aufbau von Statements ist gegeben und Nutzer kann nur Parameter festlegen
2. Gespeicherte Funktionen in Datenbank
   Aufbau von Statements ist gegeben, Zugriffsfunktion aber in DB-System
3. Whitelist Input Verifizierung
   Parameter werden nochmals in Parser mit einem Switch geprüft
   (keine ungewollten Parameter können übergeben werden)

Buffer Overflow

Angreifbar: Sprache mit Memory-Management:

- Externe/Unsichere Methoden zur Datenverwaltung benutzt
- potentieller Buffer Overflow wird übersehen

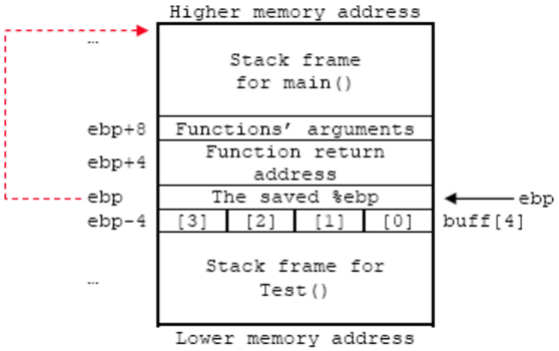
Ziel:

- Versuch, eigenen Code auszuführen
- Crash der Software

Exploit: eigenen Code in Buffer laden und per Overflow Speicherbereiche des Programms (FP/SP) überschreiben

Veranschaulichung

Normaler Call-Stack:
Argumente, Return-Address, Framepointer (ebp), Local Variables

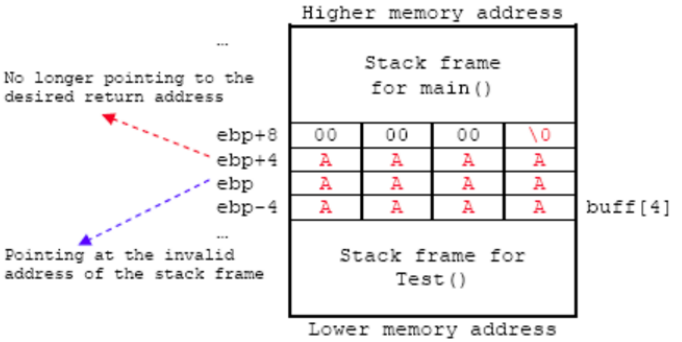


[3]

[3] Quelle: tenouk.com/Bufferoverflowc

Veranschaulichung

Call-Stack mit Overflow:
Sprung zur überschriebenen Return-Address führt zu Segmentation Fault

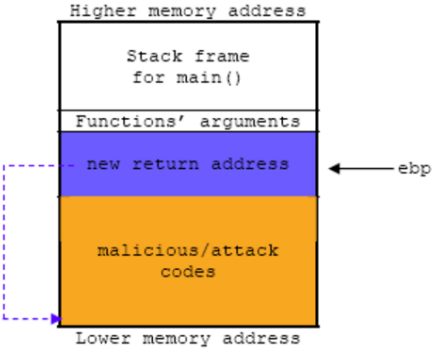


[4]

[4] Quelle: tenouk.com/Bufferoverflowc

Veranschaulichung

Call-Stack mit Injection:
Sprung zur neuen überschriebenen Return-Address (FP geht verloren)



5

[5] Quelle: tenouk.com/Bufferoverflowc

Demo - Root Shell

```
root@d2e418f1da71:/home/src# ./vulnerable $(python -c 'print "\x90"*40 + "\x
31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x
89\xe1\xb0\x0b\xcd\x80" + "A"*47 + "\x20\xce\xff\xff"')
Welcome ������������������������������������������1@P
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ``````````````````````````AAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
# whoami
root
# 
```

Problem

Kopieren eines Strings unbekannter Länge:

```
void func(char *name)
{
    char buf[100];
    strcpy(buf, name);  // unsafe
}
```

Maßnahmen

1. Wahl einer Sprache, die kein Memory-Management zulässt
   (z.B. Java, C#)
2. bei User Input auf Overflows prüfen
   (Crash evtl. unvermeidbar, aber Eindringen verhindert)
3. keine veralteten Methoden zum Lesen des Inputs verwenden
   (z.B. in C `get()` überprüft keine Ziel-Arraylänge)

Command Injection

Angreifbar: Systeme, die Input an Shell weiterleiten

Ziel: Versuch, Shell Befehle auszuführen

Exploit: Möglichkeit diese Befehle im Frontend einzufügen

Effekt: Zugriff auf System
(besonders problematisch, wenn Shell hohe Rechte hat)

Maßnahmen: Input vor Ausführung kontrollieren (ähnlich SQL-Injection)

- Feste Statements mit variablen Parametern
- Whitelist Input Verifizierung

Cross-Site Scripting

Angreifbar: Websiten

Ziel: Versuch, eigene Scripts bei einem Opfer auszuführen

Exploit: Möglichkeit diese Scripts auf dem Webserver zu hinterlegen oder von vertrauten Websiten ausgeben zu lassen

Wirkung:

- Zugriff auf Nutzerdaten
- Umleitung auf andere Website
- Installation von Schadsoftware

Arten von XSS

Stored XSS: Script auf Server gespeichert
(DB, Forum, Log, Kommentier-Bereich, etc.)
$\Rightarrow$ ausgeführt, wenn Opfer von dort Information aufruft

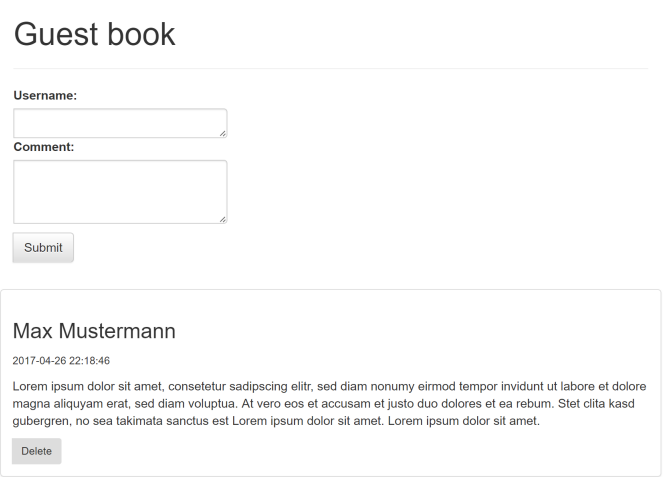
Reflected XSS: Script wird von Server ausgegeben
(Error, Suchergebnis, etc.)
$\Rightarrow$ Angreifer lenkt Opfer auf vertraute aber angreifbare Website um

Beispiel - Reflected XSS Angriff

Simple Datenbanksuche, die bei der erfolglosen Suche nach BEGRIFF eine Fehlermeldung ausgibt:
"BEGRIFF konnte nicht gefunden werden"

1. Generiere Link für eine Suche (z.B. "mydb.de/search=BEGRIFF")
2. Tausche BEGRIFF durch Schad-Script
3. Leite Opfer zu diesem Link
4. Fehlermeldung wird Script anzeigen
5. Script wird ausgeführt, da es sich um vertraute Website handelt

Demo - Gästebuch



Problem

Speichern des Kommentars:

```
<?php
if (isset($_POST["comment"]) && is_string($_POST["comment"])) {
    $comment = $_POST["comment"];

    $sql = "INSERT INTO comments(timePosted, username, comment)"
            . " VALUES ('"
            . $timestamp . "', '" . $username . "', '" . $comment
            . "')";

    // ...
}
?>
```

Maßnahmen

1. Input generell nicht in HTML-Dokument erlauben
2. Input in bestimmten Bereichen (Tags) vermeiden
   (script, comment, attribute name, tag name, CSS)
3. Input escapen, bevor es als Attribut verwendet wird
   (alle Arten von Attribut: JavaScript, JSON, HTML, CSS, URL, ...)
   $\Rightarrow$ Nutzen von fertigen Librarys zum Escapen

Cross-Site Request Forgery

Angreifbar: Websiten

Ziel: authorisiertes Opfer soll Handlung des Angreifers ausführen

Exploit: Opfer wird auf Website weitergeleitet und führt dort unwissentlich Handlung aus

Wirkung: Änderung von Eigenschaften der Website mit Rechten des Opfers
(Nutzerdaten, Inhalte, Berechtigungen, ...)

Maßnahmen:

- vor kritischen Operationen Bestätigung anfordern
- Nutzen eines Identifiers für jede HTTP-Anfrage
  (Identifier wird an Nutzer gesendet, Nutzer muss diesen zurücksenden)

Vorgang

Teilnehmer:

- authorisierter Nutzer: Norbert
- Angreifer: Albert
- Website

Vorgang

1. Norbert arbeitet an der Website - ist dazu eingeloggt
2. Albert konstruiert einen Link der die Website instruiert, einen neuen Admin Account für Albert zu erstellen
3. Albert bringt Norbert dazu, dem Link zu folgen
4. die Website erstellt Admin Account - die Anfrage kam ja von Norbert (der die Rechte dazu besitzt)

Demo - User Account System

Problem

Bearbeiten des ("Passwort ändern") Requests:

```
<?php
if(isset($_GET['changePassword'])
&& isset($_POST['passwort']) && isset($_POST['passwort2'])) {
    $passwort = $_POST['passwort'];
    $passwort2 = $_POST['passwort2'];

    if ($passwort == $passwort2) {
        // save password in db ...
    }
}
?>
```

Danke für Eure Aufmerksamkeit

6

[6] edgis-security.org/infosec/responsible-disclosure/

Quellen

- Open Web Application Security Project (www.owsap.org)
- technopedia - Software Security (www.technopedia.com)
- synopsys - What is Software Security (www.synopsys.com)
- heise c't - Exploits
- exploit-db.com
- Wikipedia
- raygun.com (raygun.com/blog/10-costly-software-errors-history/)
- www.heise.de/mac-and-i/meldung/iOS-Bug-Bounty-Programm-Exploit-Haendler-will-mehr-zahlen-als-Apple-3293301.html
- www-users.math.umn.edu/ arnold/disasters/patriot.html