Software Security

Kryptographie und IT Sicherheit SS 2018

Dmitrii Polianskii, Manuel Klappacher

Universität Salzburg

Themen

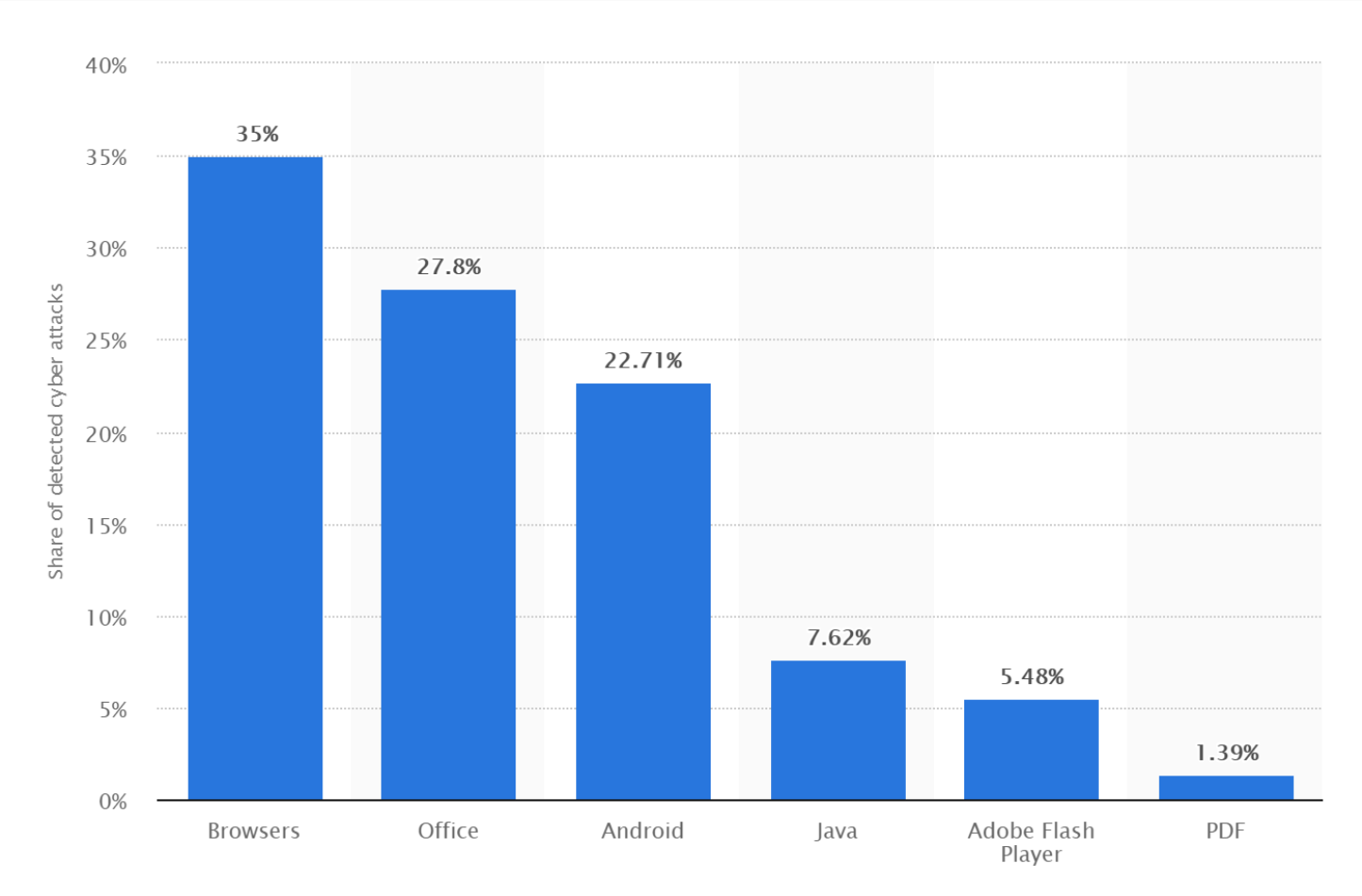
Einleitung

Wie entstehen Fehler und Sicherheitslücken?

- Programmierfehler
  - Treten sehr häufig auf
  - Logische Fehler, syntaktische Fehler, lexikalische Fehler
  - Zeitdruck
  - Mangelnde Kenntniss
  - Keine ausreichenden Tests
- Compilerfehler
  - Treten nicht sehr häufig auf
- Absichtlich platzierte Backdoors
  - Sehr schwer nachzuweisen - wie Unterscheidet man Fehler von böswilliger Absicht?
  - Werden auch von anderen Teilnehmern entdeckt und von Kriminellen dann für ihre Zwecke missbraucht
- Zuviel Komplexität
  - Komplexe Systeme kann keiner mehr überblicken
  - schwer über Sicherheit argumentierbar

Einleitung - Statistiken

Most commonly exploited applications worldwide as of 3rd quarter 2017



Quelle: www.statista.com

Open Source Vorteile

- Programmcode kann überprüft werden, Sicherheitslücken fallen leichter auf
- Erschwert implementierung von Backdoors
- Software kann von der Communtiy weiterentwickelt oder geforkt werden
- Bestimmte Funktionen können abgedreht werden

Spezialfall Firmware

- Gerät wurde bereits verkauft, kein Interesse des Herstellers an Updates
- Zu viele verschiedene Geräte - Unmöglicher Verwaltungsaufwand
  - Alleine Samsung hat bis 2014 56 verscheidene Smartphones pro Jahr herausgebracht
- Firmware agiert in Schicht unter Betriebssystem - Angriffe können vom Benutzer nicht erkannt oder verhindert werden
- Firmware meist Closed Source - keine Weiterentwicklung der Community

Sicherheitslücken in Firmware - Beispiele

- BadUSB - Eingabegeräte, USB-Sticks, Speichermedien, Kameras, ...
- Intel ME - Betriebssystem im Prozessor (AMD PSP)
  - Funktionsweise undokumentiert
  - Kritische Lücke 2017 entdeckt
  - NSA und Google haben Intel ME abgeschaltet auf ihren Geräten
- Android
  - praktisch alle Android Geräte ohne Sicherheitupdates
- Router, Smart TV's, IoT-Devices - Millionen angreifbare Geräte in Haushalten, Firmen und Behörden

Smartphones Sicherheitsupdates

Global security update availability for Smartphones
(January/February 2018 Report)

OS	Brand	Shortest time to publish a SU		Max worldwide availability delay**		SU is carrier independant for ALL devices	Support duration for security updates (2016)	Support duration for security updates (2017)		Devices SU's availability rate after 1 Month***
		For the first device	For all supported devices	Manufacturer Update	Carrier Update			Minimum	Maximum	
iOS	Apple	Day(s)	Day(s)	1 Day	-	Yes	5 years	4 years*	5 years	ALL devices
Windows	Microsoft / Nokia	Day(s)	Day(s)	1 Day	-	Yes	3 years	4 years		ALL devices
PrivatOS	Silent Circle	Weeks/Month*	N/A	1 Day	-	Yes	3 years	3 years		ALL devices
Android	Essential	Day(s)	N/A	1 Day	Month(s)*	No	N/A	3 years *(Expected)**		High
	Google	Day(s)	Day(s)	2 weeks*	Month(s)	No	2 years	3 years		High
	BlackBerry	Week(s)	Week(s)	Week(s)	Month(s)	No	2 years	2 years		Medium/High
	Nokia (HMD)	Week(s)	1 Month	Week(s)	Month(s)	No	N/A	2 years *(Expected)* *		Medium/High
	Sony	Week(s)	Month(s)	Quarter(s)	Quarter(s)	No	1,5 years	1,5 years	2 years	Medium/High
	FairPhone	Week(s)*	N/A	1 Day*	-	Yes	1,5 years*	2 years*		ALL devices but partially updated*
	Huawei	Week(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1,5 years	2,5 years	Medium/Low
	LG	Week(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1,5 years	2,5 years	Medium/Low
	Samsung	Week(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	2,5 years	Medium/Low
	Asus	Week(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	1,5 years	Low
	Motorola (Lenovo)	Week(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	2 years	Low
	OnePlus	Month(s) *	Month(s)	Quarter(s)	-	Yes	1/1,5 years	1,5 years	2 years	Low & partially updated*
	Honor (Huawei)	Month(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	1,5 years	Low
	HTC	Month(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	1,5 years	Low
	Blu (Tinno)	Month(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	1,5 years	None
	Wiko (Tinno)	Month(s)	Month(s)	Quarter(s)	Quarter(s)	No	1/1,5 years	1 year	1,5 years	None

SU = Security Update. After a high or critical security breach has been unveiled.

* Apple : They stopped supporting iPhone 5C in 2017 after 4 years, all other devices since iPhone 4S (2011) have been supported for 5 years.
* Silent Circle announcement : "Critical vulnerabilities are patched within 72 hours of detection or reportin", but January 2018 security patch was available only after a delay of 1 month.
* Essential : Most of US and Canadian carrier push update directly trom Essential, or in only fews days/weeks, but some carriers can also take months (like Telus) .
* Essential & Nokia : They started selling phones in 2017. We have indicated the official support announced.
* Google : Delay from official security policy https://support.google.com/nexus/answer/4457705
* Fairphone : Lasts updates doesn't cover all security vulnerability for January/February (Cover only 50% high-critical security vulnerability)
* Fairphone, duration for SU : FairPhone 1 had only 1,5 years of support (Until August 2015), FairPhone 2 had in 2017 2 years of support.
* OnePlus : deploy partial updates for limited high-critical security updates every month. Full security update are usually every 2 months.

Exploits

Code Injection

Code Injection ist das ausnutzen von Bugs durch Eingabe von ungewollten Parametern, um dadurch die Ausfürung zu verändern. Kann folgende Auswirkungen haben:

- Daten in SQL Tabellen verändern
- Installieren von Malware durch Server-Scripting Code zB. PHP
- Root Privilegien bekommen, durch Shell Injection oder Windows Service
- Angtiff auf Web User durch Cross-Site-Scripting in HTML/JS

Code Injection - Masnahmen

Kann erschwert werden durch:

- API's benutzen, die sicher gegenüber allen Symbolen sind, indem der Eingabestring compiliert und gefiltert wird.
- Whitelisting von erwünschten Parametern

SQL Injection

Ausnutzen von Sicherheitslücken in Zusammenhang mit SQL-Datenbanken. Ziele:

- Daten auszuspähen oder zu verändern
- Kontrolle über Server zu erhalten

SQL Injection - Beispiel

Es wird zusätzlicher Code bei Aufruf eingeschleust, der die Benutzertabelle modifiziert.

	Erwarteter Aufruf
Aufruf	`http://webserver/cgi-bin/find.cgi?ID=42`
Erzeugtes SQL	`SELECT author, subject, text FROM artikel WHERE ID=42;`
	SQL-Injection
Aufruf	`http://webserver/cgi-bin/find.cgi?ID=42;UPDATE+USER+SET+TYPE="admin"+WHERE+ID=23`
Erzeugtes SQL	`SELECT author, subject, text FROM artikel WHERE ID=42;UPDATE USER SET TYPE="admin" WHERE ID=23;`

Cross Site Scripting - XSS

Bei Cross-Site Scripting (XSS) werden Sicherheitslücken in Webanwendungen ausgenutzt um schadhaften Code in sonst vertrauenswürdigen Websiten einzubinden. Das ist der Fall wenn es nicht Vertrauenswürdigen Dritten erlaubt ist, Daten und Code hochzuladen.
Ziele:

- Benutzerkonten zu übernehmen
- Daten (Identitätsdiebstahl)

Der Browser des Benutzers kann nicht zwischen schädlichem und gewünschtem Code unterscheiden.

Cross Site Scripting - reflektierte Angriffe

Eine Benutzereingabe wird direkt vom Server wieder zurück gesendet. Wenn diese Eingabe Scriptcode enthält, die vom Browser des Nutzers interpretiert wird, kann dort Schadcode ausgefürt werden. Beispiel: Suchfunktion.

```
http://example.com/?suche=Suchbegriff

http://example.com/?suche=<script type=
      "text/javascript">alert("XSS")</script>

<p>Sie suchten nach: <script type=
      "text/javascript">alert("XSS")</script></p>
```

Ausgenutzt wird das dynamisch generierte Websiten ihren Ihnalt an übergebene Eingabewerte anpassen, durch HTTP-GET und HTTP-POST. Dieser Typ heisst auch nicht-persistent, da der Schadcode nur temprär bei der jeweiligen Generierung der Website eingeschleust wird.

Cross Site Scripting - persistente Angriffe

Unterscheidet sich von reflektierenden Angriffen nur dadurch, dass der Schadcode auf dem Server gespeichert wird, wodurch er bei jeder Anfrage asugefürt wird. Ist bei Webanwendungen möglich, die Benutzereingaben serverseitig ohne Prüfung speichern und diese später wieder ausliefert. Beispiel Posting auf Website:

```
Eine sehr gutes Produkt!<script type=
    "text/javascript">alert("XSS")</script>
```

Cross Site Scripting - DOM-basierte Angriffe

Webapplikation auf dem Server ist hier nicht beteiligt, wird auch lokales XSS genannt. Somit auch statische HTML Seiten mit JavaScript unterstützung anfällig für diesen Angriff.

Cross Site Scripting - Schutzmaßnahmen

- Anstatt Blacklist mit bösen Eingaben zu führen, besser Whitelist mit buten Eingaben. Da die Anzahl der Angriffmethoden nicht bekannt ist.
- HTML-Metazeichen durch Zeichenreferenzen ersetzen, damit sie als normale Zeichen behandelt werden
- Sicher programmierte Anwendung sind Web Apllication Firewalls (WAF) vorzuziehen.

Path Traversal Attack

Ein HTTP Angriff, bei dem ein Angreifer Zugriff auf gesperrte Verzeichnisse gewinnt und Code ausserhalb des Web Root Verzeichnisses ausführt.

Web Container Encoding:

```
..%c0%af represents ../
..%c1%9c represents ..\
```

Null bytes %00 können injeziert werden um Dateinahmen zu terminieren.

```
?file=secret.doc%00.pdf
```

Java sieht .pdf, Betriebssystem sieht .doc

Path Traversal Attack - Beispiele

Beispiel Zugriff auf Dateien:

```
http://some_site.com.br/get-files.jsp?file=report.pdf
http://some_site.com.br/some-page.asp?page=index.html
```

UNIX Passwort abgreifen:

```
http://some_site.com.br/../../../../etc/shadow
http://some_site.com.br/get-files?file=/etc/passwd
```

Auch möglich Dateien und Scripte von externen Websiten einzibinden:

```
http://some_site.com.br/some-page?page=http:
  //other-site.com.br/other-page.htm/malicius-code.php
```

Path Traversal Attack - Schutzmaßnahmen

- Nutzereingaben vermeiden wenn möglich, bei Datei System Aufrufen
- Indexes anstatt Dateinahmen verwenden, für Benutzereingaben
- Nutzer soll nicht ganzen Pfad eingeben können, mit eigenem Pfad umgeben
- Pfade normalisieren
  - "." Segmente entfernen
  - ".." - Segmente die ein nicht-".." Segment dafor haben werden entfernt
  - Wenn Pfad relative und das erste Segment enthält ein ":" Charackter dann wir ein "." vorangestellt

Format String Attack

Format Funktion ist eine ANSI C Funktion, um primitive Variablen in eine lesbare Ausgabe konvertieren. z.B. *printf*, *fprintf*

- Sind C/C++ Probleme
- treten heute nicht sehr häfig auf, da sie sich sehr leicht erkennen lassen

Ziele:

- Programmcrash
- Schadcode ausführung

Format String Attack - Beispiel I

```
int main (int argc, char **argv)
{
    char buf [100];
    int x = 1 ;

    snprintf ( buf, sizeof buf, argv [1] ) ;
    buf [ sizeof buf -1 ] = 0;

    printf (   Buffer size is: (%d)
         \nData input: %s \n    , strlen (buf) , buf ) ;

    printf (    X equals: %d/ in
         hex: %#x\nMemory address
         for x: (%p) \n    , x, x, &x) ;

    return 0 ;
}
```

Format String Attack - Beispiel II

Erwartete Eingabe:

```
./formattest    Bob
```

Ausgabe:

```
Buffer size is (3)
Data input : Bob
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)
```

Format String Attack - Beispiel III

Schwachstelle ausgenutzt, %x := Ausgabe Hexadezimal:

```
./formattest   Bob %x %x
```

Anstatt %x Wert von Bob auszugeben, gibt nun auch den Inhalt der Speicher Adresse aus:

```
Buffer size is (14)
Data input : Bob bffff 8740
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)
```

printf Argument sieht nun folgendermaßen aus:

```
printf (   Buffer size is: (%d) \n Data input:
    Bob %x %x \n    , strlen (buf) , buf ) ;
```

Buffer Overflows

Durch Programmfehler werden zu große Datenmengen in einen zu klein reservierten Speicherbereich geschrieben. (Buffer oder Stack, auch Pointer).

$\rightarrow$ Daten werden überschrieben:

- Schadcode wird ausgeführt
- Absturz des Programms
- Beschädigung oder Verfälschung von Daten

Zum Beispiel die Rücksprungadresse eines Unterprogrammes wird überschrieben.

Buffer Overflows

Begünstigt durch Van Neumann Architektur, Daten und Programm im selben Speicher.

- Compilierte und assemblierte Sprachen anfällig
- Anfällige Sprachen, z.B. C/C++
- Unsichere Libraries in C/C++
- Unsicheres Behandeln von Stirngs und Arraygrößen

Schutzmaßnahmen:

- Type-Safe Programmiersprachen verwenden, welche Memory Management zB Java, Python, Ruby,...
- Überprüfen auf Overflows bei User Eingaben
- in C sichere Methoden verwenden, *get_s* anstatt *get*.

Buffer Overflows - Type-Safe Sprachen

Compiler stellt Typsicherheit her, indem Datentypen geprüft werden, damit keine Typverletzungen entstehen. Wenn Typverletzungen spätestens zur Laufzeit erkannt werden, spricht man von Typsicheren Programmiersprachen.

Beispiel String in Python, es reicht der Variable einen String zuzuweisen.

```
mystring = "This is my string"
```

Beispiel in C, es muss der Typ deklariert und auch der Speicher manuell reserviert werden.

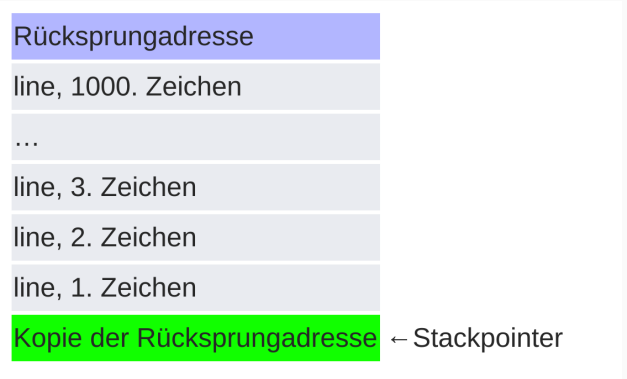
```
char mystring[20] = "This is my string";
```

Wenn man in C nun einen 30 Byte String zuweist entsteht eine Overflow Situation.

Buffer Overflows - Compiler Maßnahmen

Moderne Compiler wie neue Versionen des GNU C-Compilers erlauben die Aktivierung von Überprüfungscode-Erzeugung bei der Übersetzung.

- Zufallsvariable erstellt und überprüft, bei Veränderung wurde auch die RA überschrieben.
- Kopie der Rücksprungadresse wird unterhalb lokaler Variablen abgelegt.



Buffer Overflows - Programmbeispiel

reserved frame

Stack Overflows

Die Rücksprungadresse eines Unterprogramms und dessen lokale Variablen werden auf einen als Stack bezeichneten Bereich zu gelegt.

```
void input_line()
{
  char line[1000];
  if (gets(line))
    puts(line);
}
```

Rücksprungadresse	
1000. Zeichen	
… …	
3. Zeichen	
2. Zeichen	
1. Zeichen	←Stackpointer

modifizierte Rücksprungadresse	
line, 1000. Zeichen	
…	
line, 5. Zeichen	drittes Byte im Code
line, 4. Zeichen	zweites Byte im Code
line, 3. Zeichen	Ziel der Rücksprungadresse, Programmcodestart
line, 2. Zeichen	
line, 1. Zeichen	←Stackpointer

Buffer Overflows - Heap Overflows

Ist ein Buffer Overflow, der im Heap Bereich stattfindet.

- Daten werden zur Laufzeit gespeichert (malloc)
- Kein Limit, ausser RAM Größe
- in iOS Jailbreaks verwenden Heap Overflows um Code in den Kernel zu injizieren

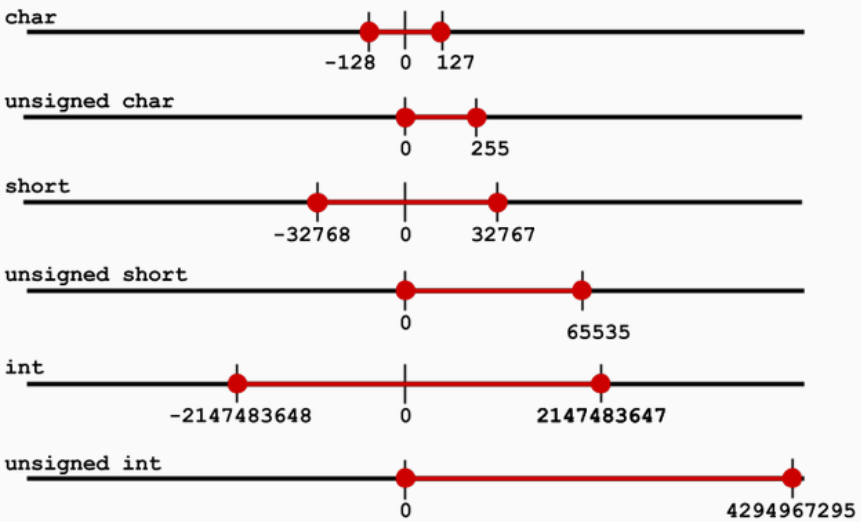
Gegenmaßnahmen:

- Code und Daten trennen mit Prozessoren - NX-bit - No Execute Bit
- Betriebssysteme mit ASLR - Address Space Layout Randomization
- Checks im Heap Manager

Integer Overflows

Entstehen wenn Operationen auf Integer die maximale Größe überschreiten. z.B. arithemetische oder cast Operationen.

- testen ob Maximaler Wert überschritten ist
- Typen beachten, signed unsigned
- muss von Hand gemacht werden, keine nativen Methoden in Programmiersprachen
- in Java BigInt verwenden



Quellen

- wikipedia.org
- owasp.org
- https://docs.oracle.com/javase/7/docs/api/java/net/URI.html#normalize()
- https://www.statista.com/statistics/434880/cyber-crime-exploits/

Vielen Dank für ihre Aufmerksamkeit!