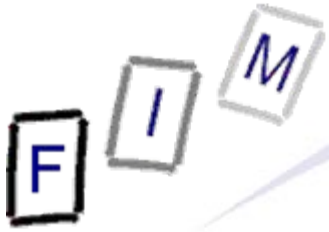
Mag. iur. Dr. techn. Michael Sonntag

Code signing

Institute for Information Processing and
Microprocessor Technology (FIM)
Johannes Kepler University Linz, Austria

E-Mail: sonntag@fim.uni-linz.ac.at
http://www.fim.uni-linz.ac.at/staff/sonntag.htm



Agenda

- Code signing overview
- Signing .NET code
  - → Strong names
  - → Authenticode
- Signing applets
  - → Java Web start

Code signing: Why?

- Typically there is only a single incentive for signing code
  - → To get it to run!
- Why?
  - → Security precautions prevent unsigned code from running
- Other reasons:
  - → Verifying integrity (viruses) etc. → More secure than hashes
  - → Preventing modifications (normal end users / attackers)
  - → Marking ownership of the code
- Problem: Signed code is not any more secure!
  - → Signature = Who “authorized” the code
  - → Signature ≠ Who “checked” the code
  - → Guarantees based on the certificate are very weak
    - » The company/person it was issued to exists
      - – Additionally sometimes: And has pledged to not distribute malware or viruses knowingly or when he should have known

Code signing: Why?

- Code signing = Authentication + Integrity
- Practice: To make sure the “program” arriving at the client actually is identical to the one produced by the author
  - → Download secured by hashes: Modify the webpage to in exactly the same way as the download to get “correct” ones
  - → Download secured by signature: You need to obtain the (typically stored offline/on other servers) stored private key
- What do you **not** get by code signing?
  - → Security guarantees, insurance, …
  - → Bug-free software
  - → Protection against decompilation
  - → Protection against modifications by user
    - » Typically the signature can be removed and the program then runs also (if security is configured appropriately!)

Bruce Schneier on code signing

- First, users have no idea how to decide if a particular signer is trusted or not.
- Second, just because a component is signed doesn't mean that it is safe.
- Third, just because two components are individually signed does not mean that using them together is safe; lots of accidental harmful interactions can be exploited.
- Fourth, "safe" is not an all-or-nothing thing; there are degrees of safety.
- And fifth, the fact that the evidence of attack (the signature on the code) is stored on the computer under attack is mostly useless: The attacker could delete or modify the signature during the attack, or simply reformat the drive where the signature is stored.

Bruce Schneier: Secrets and Lies - Digital Security in a Networked World, John Wiley and Sons, 2000

Strong names

- Applies to .NET platform: Signing assemblies
  - → There used to uniquely identify each assembly
  - → They are not intended for security
    - » They can be removed from an executable program, which will then still be able to run fine!
      - – But only with additional security configuration
  - → Additional feature: Versioning
    - » Not directly by the signature, but the associated metadata
      - – To get out of “DLL hell”: DLLs with same name but different content
- When using the Global Assembly Cache (GAC) strong names are mandatory
  - → For collision protection, not for authentication!
- Problem: Revocation of keys is not supported
- Advantages:
  - → No official certificates needed
  - → Can run offline: No online checks needed; but see revocation!

Strong Names

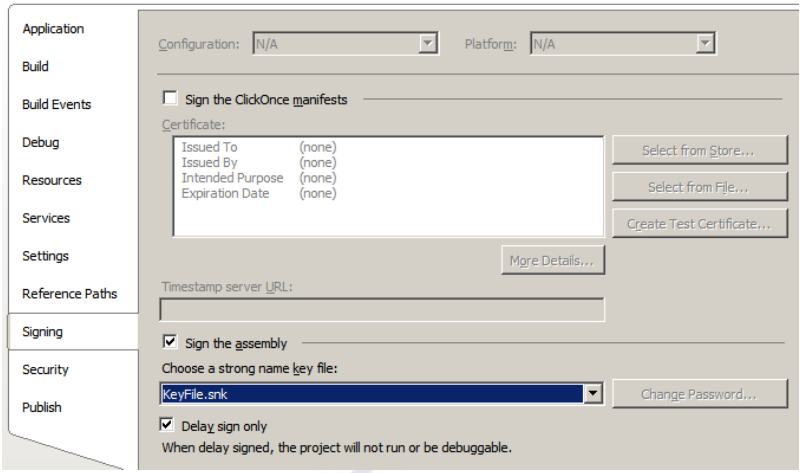
- Strong name (SN) =
  - → Text name of the assembly
  - → Version number
  - → Culture information (optional)
  - → Public key + signature
- Assemblies with SN can only reference SN-assemblies
- SN does not involve certificates, only public/private keys
  - → Referencing another assembly → Public key of that assembly is stored in the calling assembly
    - » Check at runtime whether this key is the same as the one used to sign the assembly found on disk
    - » Check whether the signature on that assembly is correct
  - → Public key distribution needed
- Since .NETv4 not really a security measure any more
  - → Integrity is still important

Strong Names
Delay signing

- Management problem:
  - → Strong signing must keep the private key absolutely secret
  - → But it must be applied every time the source code is compiled
- Solution: Delay signing
  - → Compilation is possible with the public key alone
    - » This can be distributed to all developers
  - → Must be specified in the assembly information file
    - » Compiler leaves place empty for the actual signature
  - → Actual signing takes place with another (test) key
  - → Verification must be switched off if using the GAC
    - » This is necessary on the developer machines only!
    - » Can be done on a per-assembly basis
- Attention: Before shipping signing with the “real” private key must take place!
  - → This will insert the signature into the place reserved for it

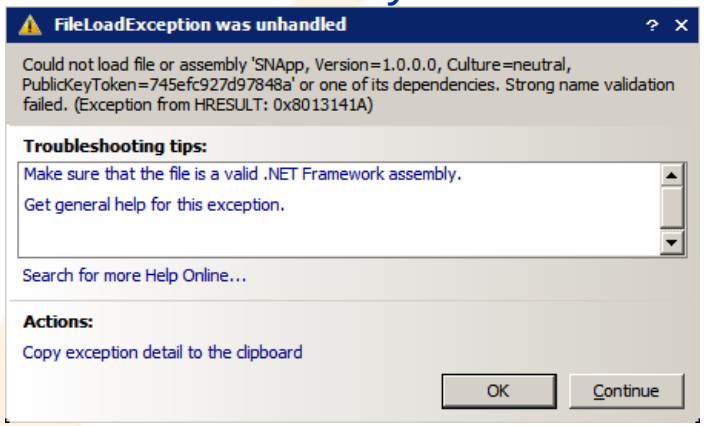
Signing code with SN

- Creating a new keypair
  - → sn –k KeyFile.snk
    - » Note: No certificate, no name, encryption, …
    - » Protection must be organized by yourself!
- Configure Visual Studio to (delay) sign the executable
  - → Take the warning seriously!
- Delay signing is more complex
  - → You need a second key pair
  - → Public key from “original”
  - → Signatur from alternative
  - → Security configuration
    to accept the alternative key (must be run as administrator!)
  - → Replaying the temporary signature before release
- We will skip the intermediate steps here!



Signing code with SN

- Run the delay-signed executable
  - → It crashes – Investigate what the real problem is
    - » The real problem is in the details: Exception Code: e0434f4d
      - – Very difficult to find out; but when debugging it:



- Apply the “real” signature: sn –R SNApp KeyFile.snk
- Now it runs!

- Verifying the signature (without running it, e.g. DLLs):
  - → sn -v SNApp.exe

Authenticode

- Uses a full certificate → As opposed to strong names the key distribution/verification becomes easier
  - → Also supports revocation checking
- Aims of Authenticode:
  - → Identifying the publisher
    - » Separation between commercial/individual users' certificates
  - → Ensuring integrity
- Signing a file does:
  - → Add the actual signature to the file
  - → Add the certificate
  - → Optionally add a timestamp (should always be done!)
    - » Requires a timestamping server; can also be added later
    - » To ensure the software can still be used when the certificate has expired (valid only for one year – "tax" on SW developers!)
    - » Revocation check for this is off by default!

Authenticode: Certificates

- Requirements for certificates
  - → Applicants must provide proof for their identity
    - » Standard certificate practice
    - » Seems to be much more relaxed regarding individuals
  - → Applicants must pledge that they will not distribute software that they know, or should have known, contains viruses or would otherwise harm a user's computer or code
  - → Commercial applicants need additionally:
    - » Minimal financial standing: DUNS number
      - – Dun & Bradstreet – a credit rating company
- Certificate is special for software publishing
  - → Actually a standard certificate with special usage restrictions
- Attention: Microsoft does NOT provide certificates!
  - → Use the “normal” certification authorities

Responsibilities of a CA

- As a leading Digital Certificate Authority, Comodo has the following responsibilities:
  - → Publishing the criteria for granting, revoking, and managing certificates
  - → Granting certificates to applicants who meet the published criteria
  - → Managing certificates (for example, enrolling, renewing, and revoking them)
  - → Storing Comodo's root keys in an exceptionally secure manner
  - → Verifying evidence submitted by applicants
  - → Providing tools for enrollment
  - → Accepting the liability associated with these responsibilities
  - → Time stamping a digital signature
- Source: http://www.instantssl.com/code-signing/code-signing-technical.html
  - → Certificates are valid for 1-3 years and cost ≈ € 170/year
    - » Plus cost of official translation of documents!

Creating an Authenticode certificate

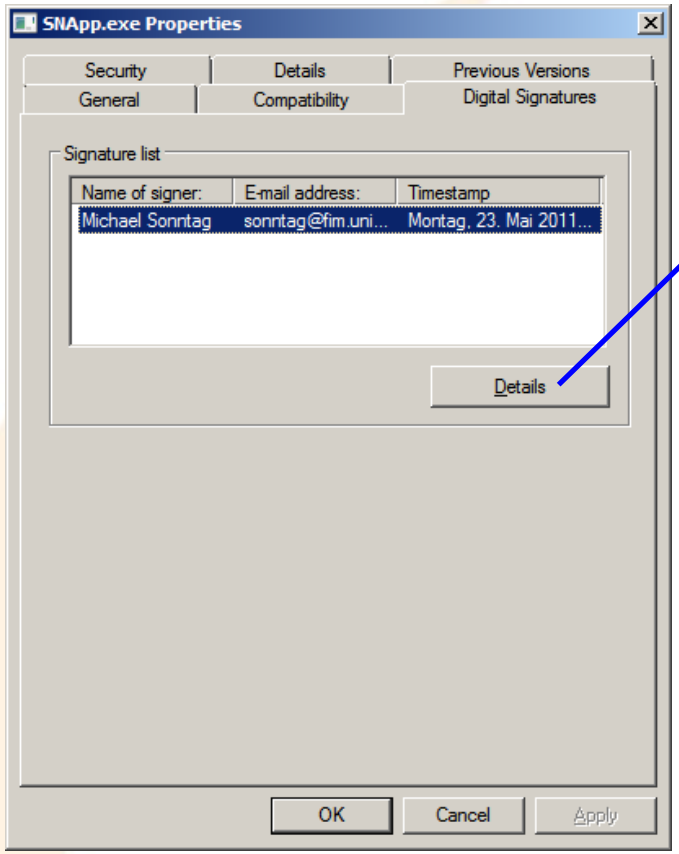
- Creating a certificate:
  - → makecert -# ! -$ individual -n "CN=Michael Sonntag,E=sonntag@fim.uni-linz.ac.at" -e 12/31/2015 -sv cert.pvk -r cert.cer
    - » Serial number: 1
    - » For individual SW publisher (alternative: commercial)
    - » Issuer & Subject: “Michael Sonntag” as Common Name
      - – And “sonntag@fim.uni-linz.ac.at” as E-Mail address
    - » End date: 31.12.2015
    - » Self-signed (“-r”)
    - » Enter (+ confirm + enter for signing) and remember the password for the private key (or enter nothing for unprotected!)
- Create a PKCS#7 object (=list of all certificates)
  - → cert2spc cert.cer cert.spc
    - » Here only one, otherwise the whole chain to the root certificate!

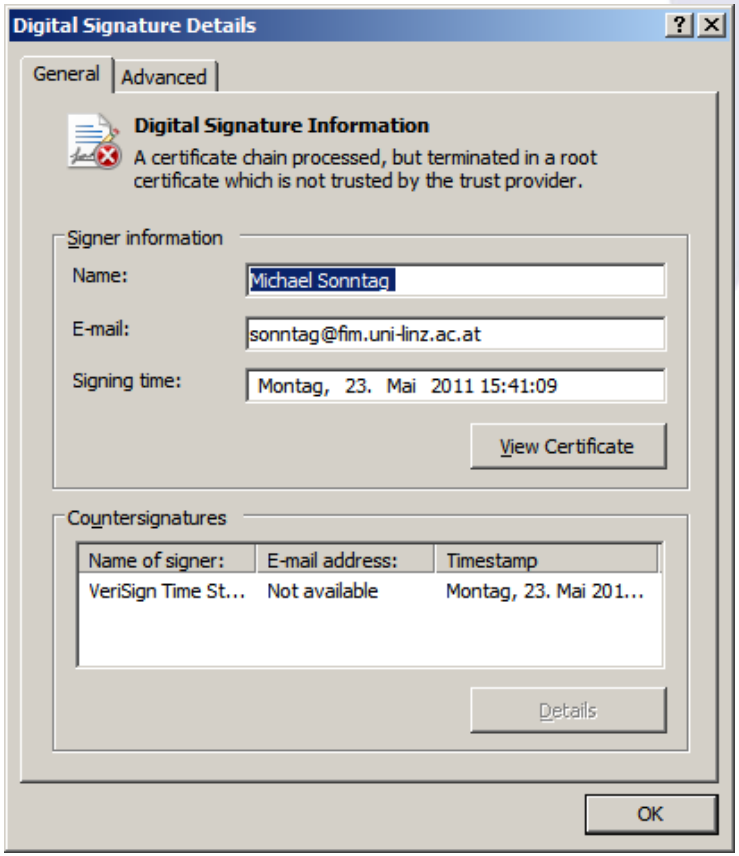
Signing code with Authenticode

- Combine certificate and private key
  - → pvk2pfx -pvk cert.pvk -spc cert.spc -pfx cert.pfx
- Actual signing
  - → signtool sign /d "iWwrite App" /du "http://www.iwrite.app/" /f cert.pfx /t http://timestamp.verisign.com/scripts/timstamp.dll SNApp.exe
  - → Additional information (optional!)
    - » Nice name for software
    - » URL of the developer
    - » Not verified, just for displaying
  - → Timestamp it

Verifying Authenticode

- Through the Windows Explorer
  - → Once signed, right-click shows new tab “Digital Signatures”





  - → Problem only because the certificate is self-signed and not imported into the trusted root certificates store!

Verifying Authenticode

- Programmatically:
  - → Signtool verify /r "Michael Sonntag" /tw /pa SNApp.exe
    - » Check the name in the certificate
    - » Check the timestamp
    - » Use the default authentication verification policy
      - – Otherwise it would be verified as a driver!
    - » Adding “/v” prints the certificate(s) included
- Output here:
  - → SignTool Error: A certificate chain processed, but terminated in a root certificate which is not trusted by the trust provider.
    SignTool Error: File not valid: SNApp.exe
    Number of errors: 1
  - → Note: The application can be executed perfectly and works!
- After importing the certificate as a trusted root certificate:
  - → Successfully verified: SNApp.exe

SmartScreen and code signing

- IE 9 has a new application reputation feature
  - → Downloads receive a reputation rating based on:
    - » Antivirus result, download traffic, download history, URL reputation, Windows logo (expensive!)
    - » File identifier (hash) & publisher (dig. signed) are sent to a cloud service, which stored the data and returns a reputation value
  - → Often downloaded & few complaints → Good reputation
  - → Bad reputation is fed back to the signer's certificate and from there to all other programs signed with the same certificate
- Problems:
  - → Every new version of a program has its own reputation
    - » Problem for applications changing (e.g. updated) frequently
  - → Very expensive to "get around": official certificate + logo
  - → Drawback for smaller companies/free software
  - → Digital signature alone is insufficient for "no warning"

Signing applets

- Applets run within a sandbox, prohibiting most interesting actions because of associated security dangers
- Allowing them access requires explicit permission
  - → This is possible “generally”, i.e. for all applets
  - → Or based on the signer of the applet
    - » Requiring, of course, that the applet is signed
- Problems:
  - → Configuration! The browser/applet viewer doesn’t ask, it merely allows access or blocks it!
    - » New versions: Improvements (see below)!

“New” applet security model

- All unsigned applets run within the sandbox
  - → With all locally defined exceptions
- “usePolicy” defined within the local policy file?
    - » Can be defined according to the source of the code or generally
      - – grant { permission java.lang.RuntimePermission "usePolicy"; };
  - → Yes: Signed applets receive those permissions specified in the local policy file without any user intervention
    - » These can be very fine-grained and be based on the source of the code and its signer
  - → No: Dialog asking whether to grant all permissions or not
    - » No restriction possible: Nothing or “AllPermission” only!
    - » But: For this signer and for this session only, or for all applets from this signer in the future
    - » But: Everything in the local policy is applied regardless of the user’s answer in addition!
      - – User denied access, but allowed according to local policy → Works!

“New” applet security model

- Recommendations for configuration:
  - → In companies, add a central policy file
    - » One line in the local policy file pointing to a central file on a web server which will be incorporated
  - → Two applets:
    - » One signed applet (=showing the dialog), which then modifies the policy file
    - » Another applet performing the actual function

Signing applets

- Example: Trivial applet writing to the file “C:\Temp\temp.txt” in the applet initialization (=no UI at all)
  - → Writing to a local file → Forbidden within the sandbox
  - → Executing it directly leads to an AccessControlException
  - → Remedy: Sign it!
- Generating a keypair/certificate request
  - → keytool –genkey –keystore keystore.jks –alias MyStore –dname „CN=Michael Sonntag” –validity 365
    - » Automatically generates a self-signed certificate too
- Sign the jar file
  - → jarsigner –keystore keystore.jks file.jar MyStore
- Programmatically verifying the signature
  - → jarsigner -verify -verbose -certs WriteFileApplet.jar
    - » Prints detailed information and certificate as well

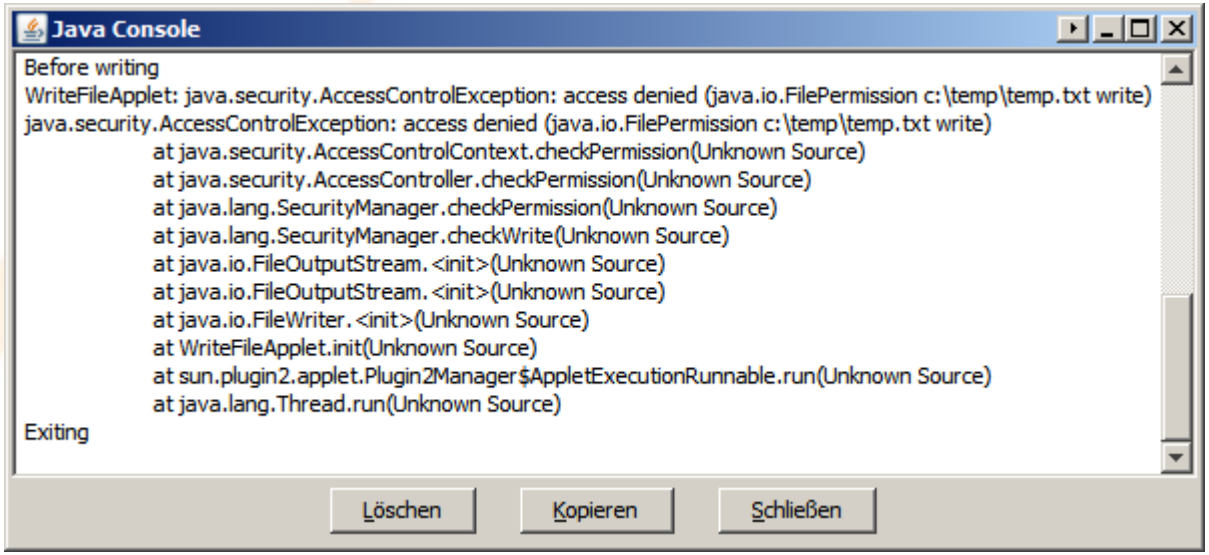
Signing applets: Result

- Creates signature file within META-INF directory inside jar
  - → Signature-Version: 1.0
    SHA1-Digest-Manifest-Main-Attributes:
    K1IZiGg6aKM/FiKTQ9VNYsurfKo=
    Created-By: 1.6.0_18 (Sun Microsystems Inc.)
    SHA1-Digest-Manifest: 3gMOg2eEQl2vQz9/G8yK1fiADRE=

    Name: WriteFileApplet.class
    SHA1-Digest: lnzY0hcvs8iwXFmIUIW/phbbLmQ=
- Adds digest values to the manifest (MYSTORE.SF)
  - → Name: WriteFileApplet.class
    SHA1-Digest: 1s95HHStGBJY8tvSqxXQGbjj50c=
- Adds binary representation of signature and certificate (MYSTORE.DSA)

Running a signed applet

- This doesn't help at all at the moment:



- What is missing are matching permission
  - → These must be administered locally
  - → There is no real user interface for it
    - » Only a tool for manipulating the policy files, but not for "installing" a policy or managing them
  - → This is a text file within the JRE path!
    - » Or specified explicitly when starting the application/applet

Creating a policy file

- Example of a separate policy file allowing only the minimum needed for this applet: Writing to a single file
  - → keystore "keystore.jks", "jks";
    grant SignedBy „MyStore" {
    permission java.io.FilePermission "c:\\temp\\temp.txt", "write"; };
- Attention: Many pitfalls!
  - → The URL of the keystore must be exactly right (no warning!)
    - » If a “file://” URL: Must use forward slashes (“/”)
  - → The file permission must use backslashes (=local name)!
  - → “SignedBy” uses the local alias in the keystore, not the name within the certificate!
  - → May also be added to the system-wide policy file
- Example:
  - → appletviewer -J-Djava.security.policy=java.policy Applet.jar
    - » “java.policy” = Filename of the policy file (see above)

Java Web Start

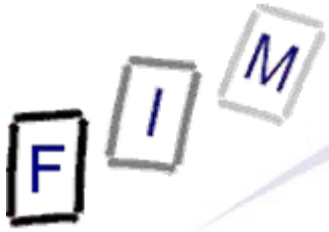
- "Distribution system" for Java applications
  - → They can be started from a web browser (downloaded only once and cached), but they don't need one
    - » They are real applications
  - → Applets can run inside JWS, then they don't need a browser
  - → JWS apps are cached indefinitely on the client and run without any network connection
    - » Automatic update check, iff network connection exists
  - → Can automatically download a specific JRE version if needed
- Reference implementation of the JNLP
  - → Java Network Launching Protocol
  - → Defines an XML schema how to start such an application
    - » Where to find jars, security configuration, update settings, …
  - → Special compression ("Pack200") to reduce jar size
- Doesn't seem to be widely used

Java Web Start

- Security: Unsigned JWS apps runs in a sandbox
  - → Some slight modifications from applet sandbox
    - » Can import/export files, print, open socket connections:
      - – After requesting user permission!
  - → Signing is identical to applets
  - → Signed JWS: No sandbox → Can do everything it wants
    - » Specific security configuration exists, but the only element currently specified is “all-permissions”!
- Implementation considerations:
  - → All jars in a JWS package must be signed with the same certificate: Unpack + re-sign them or use several JNLP files
  - → Web server must serve JWS apps with MIME type “application/x-java-jnlp-file”
    - » Browser must be configured to run this MIME type correctly
    - » Similar: *.jnlp must be associated to javaws.exe for local files
    - » Both is done by the JRE installer

Conclusions

- Code signing is difficult to get right
  - → Extensive testing needs to ensure that it works and that really no warning signs pop up
- It gives only limited advantages
  - → No warning signs
  - → No modification in transit
    - » If users can identify the publisher to be the correct one!
  - → Drivers must be signed in newer versions of Windows
- But there are shortcomings
  - → Limited to certain file types
  - → Verification is limited to specific circumstances
- Full automation in the build process is possible
  - → And highly desirable!



Questions?

Thank you for your attention!

Literature/Links

- Microsoft: Introduction to code signing
  http://msdn.microsoft.com/en-us/library/ms537361%28v=vs.85%29.aspx
- IEBlog: SmartScreen Application Reputation – Building Reputation
  http://blogs.msdn.com/b/ie/archive/2011/03/22/smartscreen-174-application-reputation-building-reputation.aspx
- Oracle: Applet Security Basics
  http://download.oracle.com/javase/6/docs/technotes/guides/plugin/developer_guide/security.html