JavaOne™

ORACLE®

Safer and Faster

New JDK Security Features and Performance Improvements

Sean Mullan
Consulting Member of Technical Staff
Oracle
October 28, 2015
@seanjmullan

Safe Harbor Statement

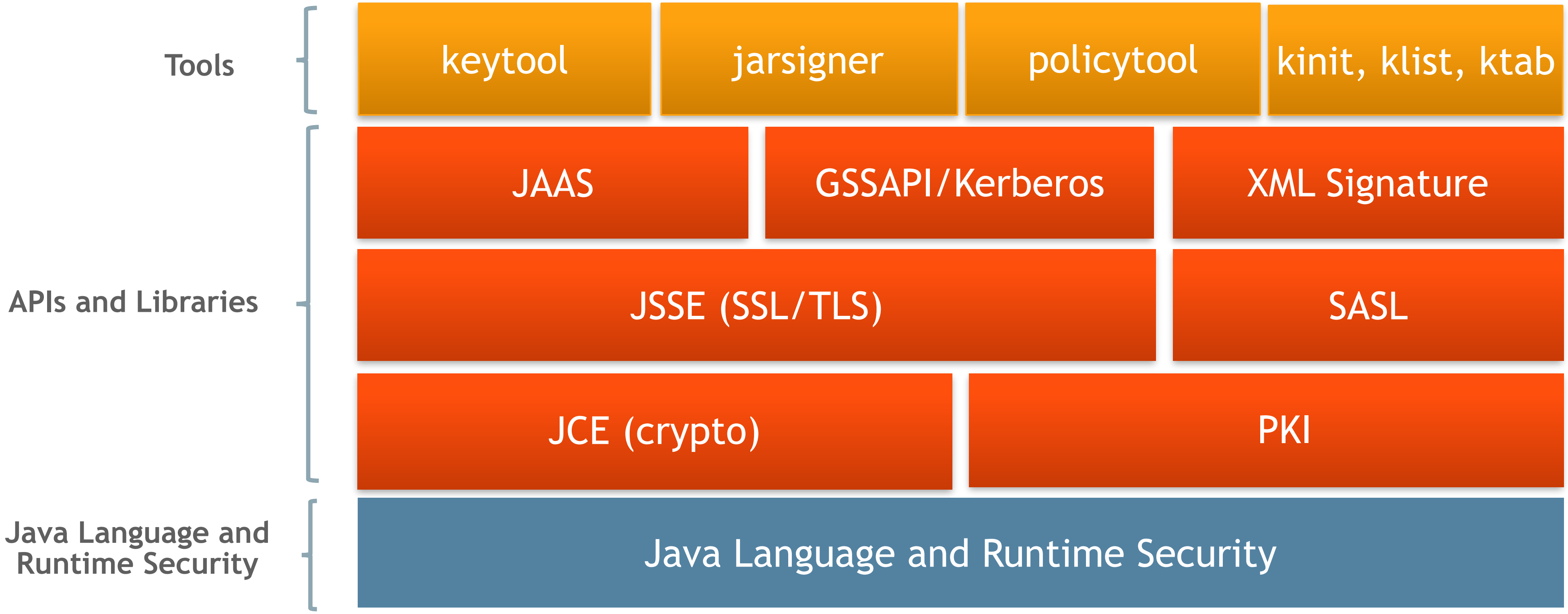
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

1. Overview
2. Secure By Default Improvements
3. Performance Improvements
4. JDK 9 Security Features
5. Conclusion

Agenda

Java SE Security Components



Agenda

What is “Secure by Default”?

- Wikipedia: *“Security by default, in software, means that the default configuration settings are the most secure settings possible, which are not necessarily the most user friendly settings.”* [1]
- OWASP: *“… by default, the experience should be secure, and it should be up to the user to reduce their security – if they are allowed.”* [2]

[1] https://en.wikipedia.org/wiki/Secure_by_default
[2] https://www.owasp.org/index.php/Establish_secure_defaults

JavaOne ORACLE

Also …

- Wikipedia: *"In many cases, security and user friendliness are evaluated based on both risk analysis and usability tests. This leads to the discussion of what the most secure settings actually are. As a result, the precise meaning of "secure by default" remains undefined."*
- OWASP: *"It is important to understand that by no means does "Secure Defaults" mean turning off all possible network applications or sockets and services. And neither do Secure Defaults mean a 100% secure environment. But, they should ensure the least number of possible loopholes and fewer drawbacks."*

Challenges

- Compatibility
- Interoperability
- Usability
- One size does not fit all
- Phased approach
- Incorporate ability to quickly adapt to changes
  - New vulnerabilities
  - Weak or broken algorithms

Java Secure Defaults

- Initial focus: provide protection against **untrusted** code
  - Default sandbox policy
  - Restrict access to internal packages
- As scope of security APIs expanded, new defaults became necessary
  - Cryptographic algorithms were weakening or being broken
  - TLS was increasingly under attack
- Going forward, module system will introduce additional safeguards
  - Not just for code running with a Security Manager

Cryptographic Algorithm Defaults

- Direct access to crypto algorithms (via JCE) is not restricted
  - Ensures long-term compatibility
    - You can still use MD2, but you do so at your own risk
- Required algorithms are evaluated at each major release
  - Ensures every SE implementation supports industry-recommended algorithms
- Some APIs have defaults
  - `KeyStore.getDefaultType()`
    - In JDK 9, default changed from `jks` to `pkcs12`
  - `SecureRandom.getInstanceStrong()`
    - Reads `securerandom.strongAlgorithms` security property (value varies per platform)

PKI and TLS Algorithm Restrictions

- Three security properties control algorithm and key size restrictions (values shown below are for JDK 8u65)
  - **`jdk.certpath.disabledAlgorithms`**`=MD2, RSA keySize < 1024`
    - Certificate validation will fail if any of the listed algorithms or key sizes are used. Includes CRLs and OCSP responses.
  - **`jdk.tls.disabledAlgorithms`**`=SSLv3, RC4, DH keySize < 768`
    - Protocols, algorithms and key sizes listed will not be negotiated in SSL/TLS sessions
  - **`jdk.tls.legacyAlgorithms`**`=K_NULL, C_NULL, M_NULL, \`

```
    DHE_DSS_EXPORT, DHE_RSA_EXPORT, DH_anon_EXPORT, \
    DH_DSS_EXPORT, DH_RSA_EXPORT, RSA_EXPORT, \
    DH_anon, ECDH_anon, RC4_128, RC4_40, DES_CBC, DES40_CBC
```

    - Algorithms and cipher suites listed will be negotiated only if enabled and there is no other alternative

Restricted Algorithm Matrix (JDK 8, 9)

Algorithm/Protocol	*CertPath*	*TLS Legacy*	*TLS Disabled*	*Notes*
MD2	√			
MD5	√ *			*** Disabled in 9, targeted to 8u71 (Jan 2016)**
SHA-1				**Plan in development**
RSA < 1024 bits	√			
DH < 768 bits	N/A		√	
DES	N/A	√		
RC4	N/A		√	
Export CipherSuites	N/A	√		
SSLv3	N/A		√	

AlgorithmConstraints API

- Applications can also use the `AlgorithmConstraints` API to implement application-specific restrictions
- Example: extending TLS to restrict certificates with SHA-1 signatures

```
SSLParameters sslParams = new SSLParameters();
sslParams.setAlgorithmConstraints(new AlgorithmConstraints() {
    @Override
    public boolean permits(Set<CryptoPrimitive> primitives,
                           String algorithm, AlgorithmParameters params) {
        return !(primitives.contains(CryptoPrimitive.SIGNATURE) &&
            algorithm.startsWith("SHA1"));
    }
    . . .
});
```

Other TLS Secure Defaults (continued)

- SSLv3 disabled by default (January 2015: JDK 8u31)
  - Added to `jdk.tls.disabledAlgorithms` security property
- RC4 Cipher Suites disabled by default in a phased approach

Other TLS Secure Defaults (continued)

- SSLv3 disabled by default (January 2015: JDK 8u31)
  - Added to `jdk.tls.disabledAlgorithms` security property
- RC4 Cipher Suites disabled by default in a phased approach
  1. Lowered position in enabled cipher suites (October 2014: JDK 8u25)

Other TLS Secure Defaults (continued)

- SSLv3 disabled by default (January 2015: JDK 8u31)
  - Added to `jdk.tls.disabledAlgorithms` security property
- RC4 Cipher Suites disabled by default in a phased approach
  1. Lowered position in enabled cipher suites (October 2014: JDK 8u25)
  2. Removed from default enabled cipher suites and added to `jdk.tls.legacyAlgorithms` property (July 2015: JDK 8u51)
     - Applications must explicitly enable RC4 using the `setEnabledCipherSuites` method of `SSLSocket` or `SSLEngine`
     - RC4 not used unless explicitly enabled and there are no other candidates

Other TLS Secure Defaults (continued)

- SSLv3 disabled by default (January 2015: JDK 8u31)
  - Added to `jdk.tls.disabledAlgorithms` security property
- RC4 Cipher Suites disabled by default in a phased approach
  1. Lowered position in enabled cipher suites (October 2014: JDK 8u25)
  2. Removed from default enabled cipher suites and added to `jdk.tls.legacyAlgorithms` property (July 2015: JDK 8u51)
     - Applications must explicitly enable RC4 using the `setEnabledCipherSuites` method of `SSLSocket` or `SSLEngine`
     - RC4 not used unless explicitly enabled and there are no other candidates
  3. Disabled by default (August 2015: JDK 8u60)
     - Added to `jdk.tls.disabledAlgorithms` property

JavaOne ORACLE

Security Tool Defaults (JDK 8, 9)

- `keytool`

Key	Key Size	Signature Algorithm
DSA	JDK 8: 1024 JDK 9: 2048	JDK 8: SHA1withDSA JDK 9: SHA256withDSA
RSA	2048	SHA256withRSA
EC	256	SHA256withECDSA

- `jarsigner`
  - digest algorithm
    - `SHA-256`
  - signature algorithm
    - same as `keytool`

JavaOne ORACLE

Module System Security Features (JDK 9)

- Strong encapsulation
  - A module’s packages must be exported for its public types to be accessible to other modules, e.g.:

```
module java.security.sasl {
    requires java.logging;
    exports javax.security.sasl;
}
```

  - Qualified exports allow you to export public types to one or more modules, e.g.:

```
module java.security.sasl {
    requires java.logging;
    exports javax.security.sasl;
    exports com.sun.security.sasl.util to jdk.security.jgss;
}
```

Module System Security Features (continued)

- Encapsulate most internal APIs (JEP 260)
  - With a few exceptions, all internal APIs will be inaccessible by default
    - Even if a Security Manager is not enabled
  - `IllegalAccessError` will be thrown if SecurityManager disabled
  - `AccessControlException` will be thrown if SecurityManager enabled

Module System Security Features (continued)

```
$ cat Test.java
import sun.security.x509.X509CertImpl;
public class Test {
   public static void main(String[] args) throws Exception {
       X509CertImpl cert = new X509CertImpl();
   }
}

$ javac Test.java
Test.java:1: error: package sun.security.x509 does not exist
import sun.security.x509.X509CertImpl;
…
$ javac -XaddExports:java.base/sun.security.x509=ALL-UNNAMED Test.java

$ java Test
Exception in thread "main" java.lang.IllegalAccessError: class Test (in module: Unnamed Module) cannot
access class sun.security.x509.X509CertImpl (in module: java.base), sun.security.x509 is not exported to
Unnamed Module
    at Test.main(Test.java:7)

$ java -Djava.security.manager Test
Exception in thread "main" java.security.AccessControlException: access denied
("java.lang.RuntimePermission" “accessClassInPackage.sun.security.x509”)
 …
```

Module System Security Features (continued)

- De-privilege modules that do not require `AllPermission`
  - Loaded with extension class loader
    - Applied to modules: `java.activation, java.annotations.common, java.corba, java.transaction, java.xml.bind, java.xml.ws, jdk.accessibility, jdk.crypto.ec, jdk.crypto.pkcs11, jdk.localedata, jdk.naming.dns, jdk.scripting.nashorn, jdk.xml.dom, jdk.zipfs`
  - Apply principle of least privilege, and only grant required permissions, e.g.:

```
grant codeBase “jrt:/jdk.crypto.ec” {
  permission java.lang.RuntimePermission "accessClassInPackage.sun.security.*";
  permission java.lang.RuntimePermission "loadLibrary.sunec";
  permission java.util.PropertyPermission "*", "read";
  permission java.security.SecurityPermission "putProviderProperty.SunEC";
  permission java.security.SecurityPermission "clearProviderProperties.SunEC";
  permission java.security.SecurityPermission "removeProviderProperty.SunEC";
};
```

Agenda

1. Overview
2. Secure By Default Improvements
3. Performance Improvements
4. JDK 9 Security Features
5. Conclusion

JavaOne ORACLE

JCE Performance Improvements

- Since JDK 8, we have implemented major performance improvements to several cryptographic algorithms
- Most of these leverage JVM **Intrinsics**

How do Intrinsics Work?

- The HotSpot JIT compiler can compile bytecode in two ways
  - Normal compilation
  - Intrinsics, which are hand-written assembly code for specific methods that are embedded in JVM code generation logic
- For some performance critical code, normal compilation is not able to generate optimal code or use platform-specific instructions
- Before compiling a method, the JIT compiler checks for an intrinsic and if defined, uses it instead
- Intrinsics allow JCE to leverage ISA-specific hardware accelerated instructions

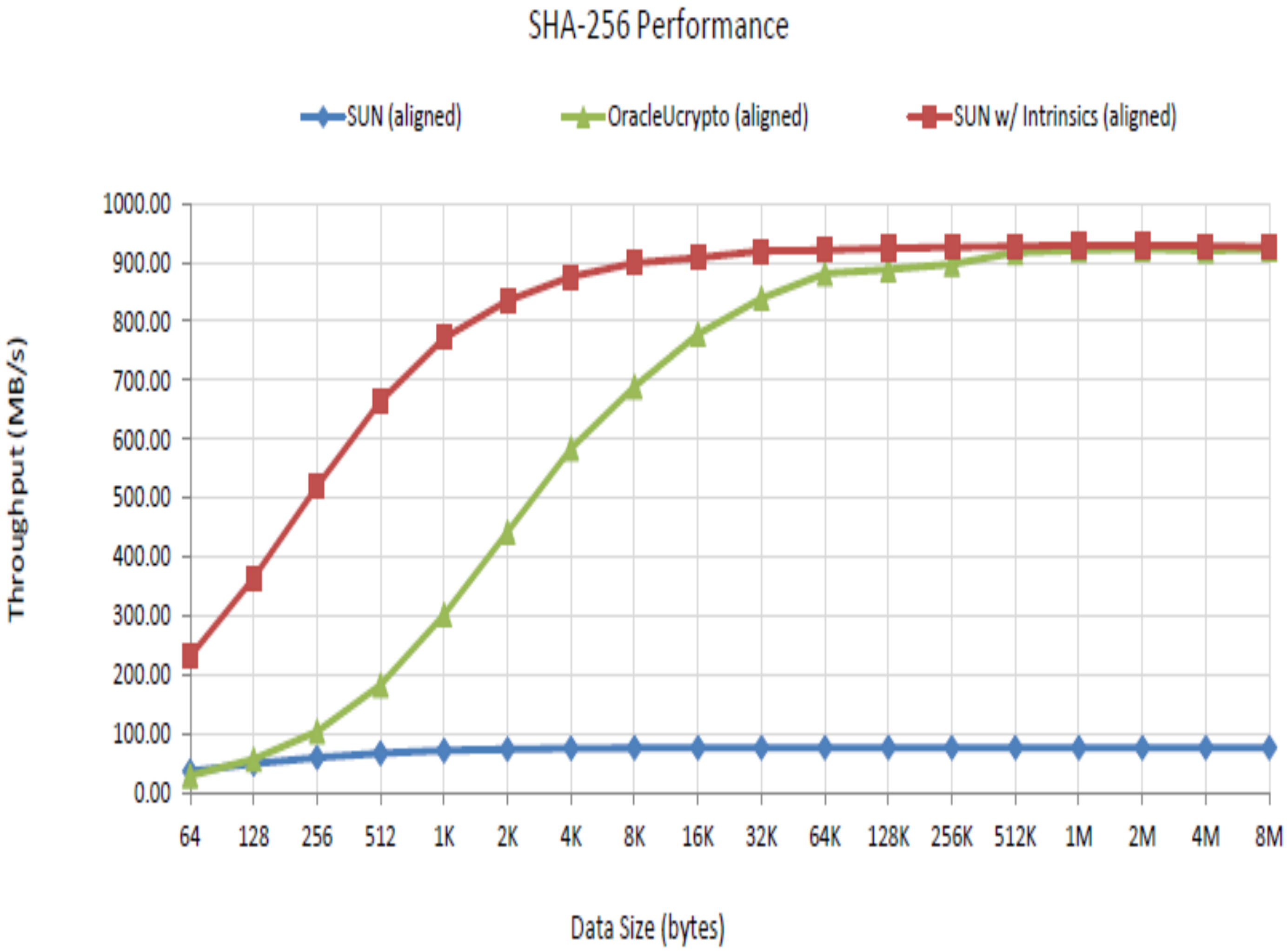
JCE Performance Improvements using Intrinsics

- JDK 8: Leverage CPU Instructions for AES Cryptography
  - http://openjdk.java.net/jeps/164
  - AES block cipher on x86 systems
- JDK 8u20: Support for AES on SPARC
  - https://bugs.openjdk.java.net/browse/JDK-8002074
  - SPARC T4 Systems and beyond

JCE Performance Improvements using Intrinsics (cont)

- JDK 8u40: Leverage CPU Instructions to Improve SHA
  - http://openjdk.java.net/jeps/207
  - SHA1 and SHA2 message digests on SPARC systems
- JDK 9: Leverage CPU Instructions for GHASH and RSA
  - http://openjdk.java.net/jeps/246
  - GHASH (used in AES/GCM mode) on x86/SPARC systems and RSA on x86_64 systems

SHA-256 Performance on SPARC



- SHA-256: up to **7.8x** faster (in throughput) than OracleUcrypto
  - More gain at smaller data sizes
- SHA-1 (not shown): up to **7.7x** faster than OracleUcrypto
- SecureRandom
  - SHA1PRNG: 48% improvement

HTTPS Benchmark Performance on SPARC

Benchmark: HTTP Servlet
Response Size = 1K
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
Number of Users: 90

Providers	Throughput (ops/sec)	%CPU	Ops/sec/ %CPU	Gain (factor)
Java w/o Intrinsics	76185	91	837	1
Java+Ucrypto	88816	82	1083	1.29
Java+AES Intrinsics	97791	76	1287	1.54
Java+AES+SHA Intrinsics	110033	66	1667	1.99

- AES and SHA Intrinsics significantly improve performance of HTTPS
- **99%** faster in throughput over pure Java

RSA Comparison with Specjvm2008 crypto.rsa on Solaris

- SPARC T7-1@4.13GHz (Tahoe)
- OS version: Solaris 12.0 b69 (March 2015)
- JDK 1.9.0 b74
- Single-threaded runs

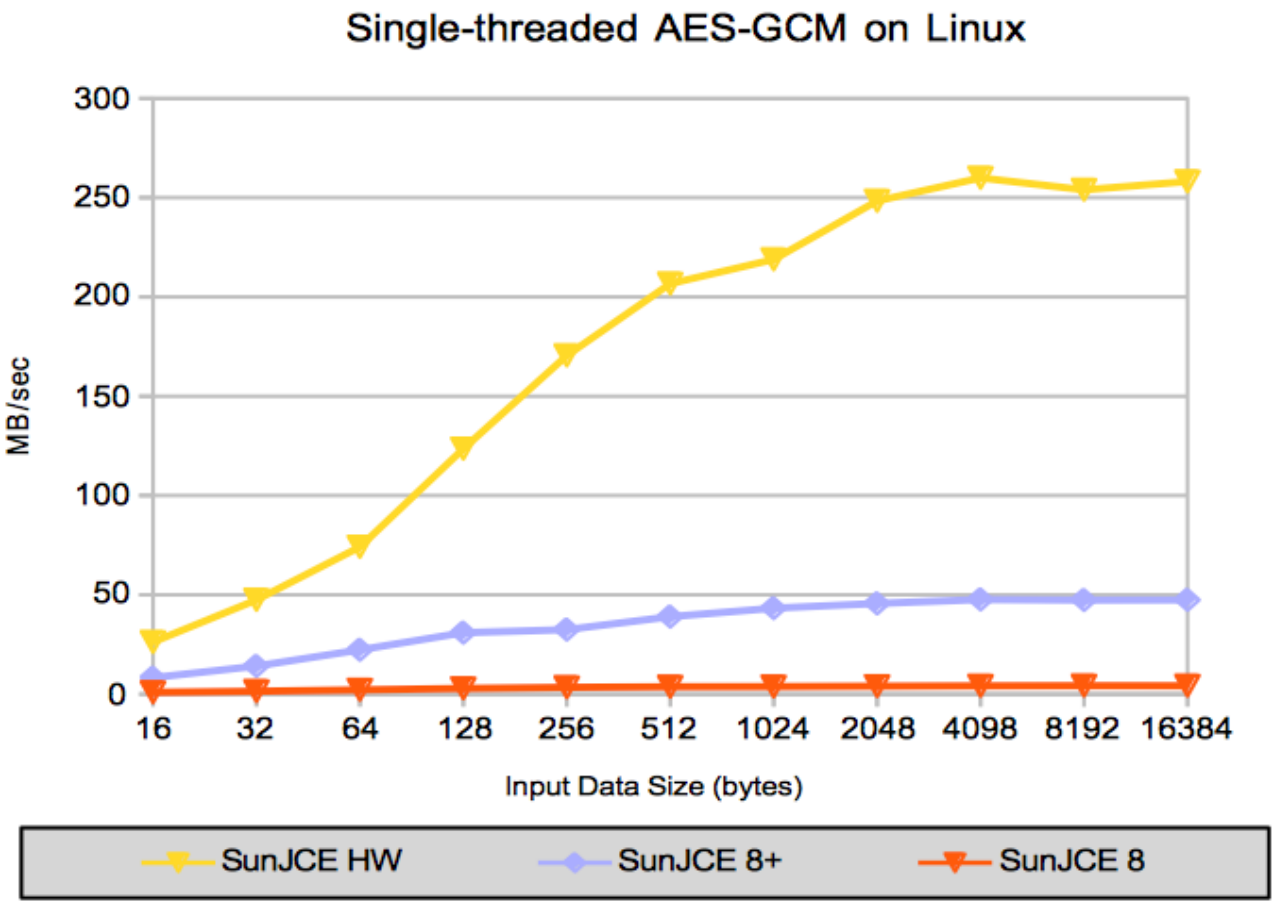
Crypto.rsa	RSA->Ucrypto	RSA->intrinsic	RSA->pure java
Throughput (Ops/min)	649.51	N/A	57.48

- Intel X5-2 Xeon E5-2690 v3 @2.60GHz (Haswell)
- OS version: Solaris 11.3 b15 (Feb 2015)
- JDK 1.9.0 b74
- Single-threaded runs

Crypto.rsa	RSA->Ucrypto	RSA->intrinsic	RSA->pure java
Throughput (Ops/min)	351.92	362.79	150.79

JavaOne ORACLE

AES/GCM Performance on x86



- JDK 9: up to a **62x** performance gain over the JDK 8 GA implementation
  - up to **5.45x** over 8u60 implementation
- 8u60 performance improved due to:
  - https://bugs.openjdk.java.net/browse/JDK-8069072

Improve Secure Application Performance (JDK 9)

- Improve performance of applications run with a Security Manager enabled (JEP 232)
- Optimizations implemented:
  - Reduced number of synchronized blocks
  - Used concurrent collections to cache Permissions, ProtectionDomains, etc
  - Improved speed of `SecurityManager.checkPackageAccess()`
  - Eliminated name service lookup from `CodeSource.hashCode()`
- Increased speed and throughput of permission checks
- Did not focus on stack walking or `AccessController.doPrivileged()`

SPECjEnterprise Benchmark Results

Method	Improvement (Inclusive CPU Time)
`AccessController.checkPermission()`	22.7%
`AccessControlContext.checkPermission()`	64%
`ProtectionDomain.implies()`	65.4%
`Permissions.implies()`	35.9%
`SecurityManager.checkPackageAccess()`	29.6%

Overall overhead of Security Manager (in response time): 4.68%

Duke wins … again!

Agenda

JDK 9 Security Features

http://openjdk.java.net/jeps/0

- JEP 219: Datagram Transport Layer Security (DTLS)
- JEP 229: Create PKCS12 Keystores by Default
- JEP 232: Improve Secure Application Performance
- JEP 244: TLS Application-Layer Protocol Negotiation Extension
- JEP 246: Leverage CPU Instructions for GHASH and RSA
- JEP 249: OCSP Stapling for TLS
- JEP 273: DRBG-Based SecureRandom Implementations

JEP 229: Create PKCS12 Keystores by Default

- Transition the default keystore type from JKS to PKCS12
- Improved security and flexibility
  - PKCS12 supports stronger cryptographic algorithms than JKS
  - PKCS12 supports secret keys and attributes
- Compatibility is maintained
  - A JKS keystore can read a PKCS12 keystore and vice-versa
- `KeyStore.getDefaultType()` will now return `“pkcs12”`
- New `KeyStore.getInstance(File, …)` methods for automatically determining type of keystore

Create PKCS12 Keystores by Default (Example)

```
// Create default keystore type
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());

// Prints “pkcs12”
System.out.println(ks.getType());

// Load keystore
try (FileInputStream fis = new FileInputStream("keystore.p12")) {
    ks.load(fis, "password".toCharArray());
}

// Or preferably, use new file probing API to automatically determine type
ks = KeyStore.getInstance(new File(“keystore.p12"), “password”.toCharArray());
```

JEP 246: Leverage CPU Instructions for GHASH/RSA

Improved Provider Configuration

- The existing provider configuration security properties (in the `java.security` file) have limitations
  - Insufficient for providers that offer large performance gains for some, but not all algorithms
    - E.g.: on Solaris, the SunJCE provider in JDK 9 offers better performance for AES/GCM but the Ucrypto provider performs better for other algorithms
- A new `jdk.security.provider.preferred` property will allow specific providers to be chosen before others, e.g.:

```
jdk.security.provider.preferred=AES/GCM:SunJCE,\
                                MessageDigest.SHA-256:SUN
```

JEP 273: DRBG-Based SecureRandom Implementations

- Implement the three Deterministic Random Bit Generator (DRBG) mechanisms described in NIST 800-90Ar1
  - Use modern algorithms as strong as SHA-512 and AES-256
  - Each can be configured to better match user requirements
  - Support for mechanisms becoming very important in some environments
- Add new methods to `SecureRandom` for DRBG operations
- Add new APIs for specifying Entropy Input as described in NIST SP 800-90B and 800-90C

DRBG-Based SecureRandom Implementations (Example)

NOTE: API is still under discussion

```
// Create SecureRandom for “HashDRBG” mechanism
SecureRandom sr = SecureRandom.getInstance(“HashDRBG”);

// Optionally, configure DRBG
// (not all code shown)
DrbgSpec spec =
    new DrbgSpec(entropyInput, “SHA-256”, 256, true, false, nonce, personal);
sr.configure(spec);

// Generate random bytes
sr.nextBytes(random);
```

JEP 219: Datagram Transport Layer Security (DTLS)

- Extend the JSSE API and implementation to support DTLS (RFCs 4347 and 6347)
- TLS must run over a reliable transport channel such as TCP
- DTLS allows applications to use TLS over an unreliable transport channel such as UDP
  - An increasing number of application protocols use UDP, e.g.: SIP, CoAP, SRTP
- Applications use the `SSLEngine` programming model to use DTLS

JEP 249: OCSP Stapling for TLS

- Implement the TLS Certificate Status Request Extensions (RFCs 6066 and 6961)
- Client-side OCSP checking incurs significant performance overhead
- With OCSP Stapling, the server is responsible for obtaining and sending the OCSPResponse to the client. This has several benefits:
  - Performance: responses can be cached and sent to all clients
  - Security
    - Allows captive portals to check revocation status
    - Avoids client-side privacy leaks

OCSP Stapling for TLS (Example)

On the Client

```
// By default, OCSP Stapling is enabled if revocation checking is enabled.
// To disable, set the system property jdk.tls.client.enableStatusRequestExtension
// to false.

SSLContext context = SSLContext.getInstance(“TLS”);
TrustManagerFactory fac = TrustManagerFactory.getInstance(“PKIX”);

// To enable revocation checking, either:
// 1. Set revocation property to true
System.setProperty(“com.sun.net.ssl.checkRevocation”, “true”);
fac.init(keyStore);

// Or, 2. use PKIXBuilderParameters and revocation is enabled by default
PKIXBuilderParameters params =
    new PKIXBuilderParameters(anchors, new X509CertSelector());
ManagerFactoryParameters trustParams = new CertPathTrustManagerParameters(params);
fac.init(trustParams);

context.init(null, fac.getTrustManagers(), null);
```

OCSP Stapling for TLS (Example)

On the Server

```
// Enable OCSP Stapling (off by default)
System.setProperty("jdk.tls.server.enableStatusRequestExtension", "true");

// Yes, that's really it!

// Optionally, several other system properties can be set for advanced usages.

// cache lifetime, in seconds (default: 3600)
System.setProperty("jdk.tls.stapling.cacheLifetime", 7200);

// cache size, number of entries (default: 256)
System.setProperty("jdk.tls.stapling.cacheSize", 128);

// and a few more …
```

JEP 244: TLS Application-Layer Negotiation Extension (ALPN)

- Add API support for the ALPN TLS Extension (RFC 7301), which provides the means to negotiate an application protocol used over a TLS connection
- An important consumer of this feature is the HTTP/2 client (JEP 110/ RFC 7540)

TLS Application-Layer Negotiation Extension (Example)

On the Client

```
SSLParameters params = sslSocket.getSSLParameters();

// Set application protocols to “h2” (HTTP/2) and “http/1.1”
params.setApplicationProtocols(new String[]{"h2", “http/1.1”});

// Optionally, set other parameters, cipher suites, etc

sslSocket.setSSLParameters(params);

sslSocket.startHandshake();

if (sslSocket.getApplicationProtocol().equals(“h2”)) {
    . . .
}
```

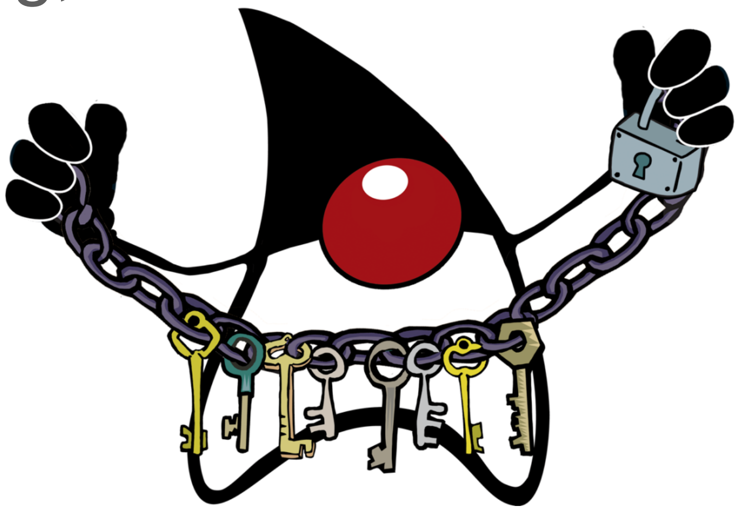
Agenda

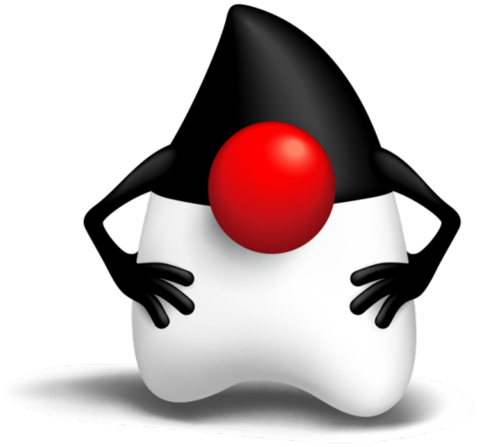
Conclusion

- Security is **important**
- We continue to make Java more secure and faster and incorporate new security features
- You can help us!
  - Let us know what you think is important
  - Participate: http://openjdk.java.net/groups/security/
  - Contribute: http://openjdk.java.net/contribute

Acknowledgements

- Solaris Performance Application Engineering Team for performance charts
- Java Security Libraries Team: Xue-Lei Fan, Frances Ho, Jamil Nimeh, Jeffrey Nisewanger, Valerie Peng, Vincent Ryan, Anthony Scarpino, Weijun Wang, Bradford Wetmore





Questions?

JavaOne™

ORACLE®