Security with VA Smalltalk

Seth Berman
Vice President of Engineering
Instantiations, Inc.
Agenda

- Security Overview
- Architectural Goals
- OpenSSL 1.1 Compatibility
- Cryptography Library
- SSL/TLS Library
Security Overview
Understanding the Value

• Secure communications is hard
  • Even Cryptographers get it wrong
  • Protocol Breakage: *SSL, PPTP*
  • Implementation Breakage: *Heartbleed*
  • Correct Protocol/Implementations can still be vulnerable
    • Side channel attacks
Security Overview
Understanding the Value

• Demand for Secure Communications
  • Is only going one way...UP!
  • Our customers are receiving increasing pressure to provide higher security applications
  • The demands extend beyond just SSL/TLS connections
Security Overview
Looking Back...

- **Before VA Smalltalk 8.6.2**
  - Dated bindings to the OpenSSL SSL/TLS library
  - No Cryptographic primitives exposed
  - Minimal help with native memory management
  - Minimal Test Cases
Security Overview
Currently...

- VA Smalltalk 8.6.2
  - Official Support for 1.0.x
  - Native Memory Management
  - Introduction of the Cryptographic Library
  - Enhanced SSL/TLS APIs
  - Test Cases (with official test vectors) for all exposed Cryptographic Algorithms
  - Story-driven code examples describing common Cryptographic Algorithm usage
Security Overview

Coming Soon…

• VA Smalltalk 8.6.3
  • Added Support for OpenSSL 1.1.0
  • Continued support for OpenSSL 1.0.x
  • Many new Cryptographic algorithms available
    • Authenticated Encryption
    • Key Derivation
  • Secure Memory Module
    • Helps protect In-Memory keys on long-running servers
Architectural Goals

- **Compatibility**
  - New Crypto layer will slide underneath SSL/TLS support
  - SSL/TLS API compatibility must be maintained
  - SSL/TLS and Crypto libraries must handle all OpenSSL versions we support
    - Currently: OpenSSL 1.0.0, 1.0.1, 1.0.2
    - Next Release: Adding OpenSSL 1.1.0
  - Differences between various OpenSSL versions should be transparent to the user (*except algorithm availability*)

- **Separation of Concerns**
- **Performance**
- **Safety**
Architectural Goals

• Compatibility

• **Separation of Concerns**
  • API Objects
    • Users should only interact with these
  • Dispatching Engine
    • Performs threaded calls
    • Error Detection and Notification
  • Native Memory Management
    • Various mechanisms to make working with native memory safer and prevent certain classes of errors.

• Performance

• Safety
Architectural Goals

- Compatibility
- Separation of Concerns
- **Performance**
  - Calls to OpenSSL are made on native threads
    - Asynchronous callouts which block only the calling ST process
  - Our thread-locking implementation plugs into OpenSSL to manage concurrency issues
  - This allows for the usage of multiple cores for higher throughput
- Safety
Architectural Goals

- Compatibility
- Separation of Concerns
- Performance
- **Safety**
  - Uses a Native Memory Manager
  - Uses a Smalltalk GC Notifier to help make sure the object’s native memory was freed
  - Various OpenSSL APIs may answer
    - Static memory (this should never be freed)
    - Reference counted memory (OpenSSL’s memory manager)
    - Unmanaged memory that the user must free
  - The Native Memory Manager keeps track of memory ownership and reference counts
OpenSSL 1.1
Overview

• Major revamp of the OpenSSL codebase
  • Post-Heartbleed: It’s getting the attention it deserves now
  • More resources applied, both internal and external
  • FIPS 140-2 Accreditation is now sponsored
• At this time: OpenSSL 1.1.0 Beta7
• With the good comes the bad...API breakage😞
OpenSSL 1.1
Hiding the API Breakage

• Version-adapting memory layout
  • All bindings to structures reconfigure their layout to meet the OpenSSL version layout specification
  • OpenSSL 1.1 uses opaque structures
    • So...we configure to those too and provide the various OpenSSL getter/setter APIs
OpenSSL 1.1
Hiding the API Breakage

• Version fallback logic
  • General OpenSSL 1.1 APIs we added implement fallback code for lower version levels
  • This was done by implementing the OpenSSL logic in Smalltalk
  • We don’t do this for algorithms as this could lead to side-channel attacks
    • Our implementation may be correct.
    • But perhaps observable cpu or caching behavior leaks information
    • Or semantics of basic primitive operations were not considered
    • i.e. trialKey = storedKey 😞 (not constant-time equality)
Cryptographic Library
Overview

• Secure Memory
• Streaming API
• Message Digests
• Message Authentication Code (MAC)
• Symmetric Ciphers
• Public/Private Key
• Key Derivation
• Secure Random Number Generator
• X509
• ASN1
Cryptographic Library
Secure Memory

- Mechanisms to secure in-memory storage
- Intended for long running servers
  - Lots of sensitive data in memory
  - This sensitive data is long-lived
  - More aggressive thread-model
- Our Secure Objects also override common APIs to expose as little as possible in case it gets logged
Cryptographic Library
Secure Memory on Linux/Unix

- **Strategy**
  - Attempt to prevent paging sensitive data to disk
  - Should not show up in a core-dump
  - Special heap should be page-guarded to protect against buffer overrun/underrun

- **Uses OpenSSL 1.1 Secure Arenas**
  - Implements the strategy above
Cryptographic Library
Secure Memory on Windows

• **Strategy**
  • Limit the time window that sensitive data could be observed in decrypted form
  • Assume paging to disk or being core-dumped is unavoidable
  • Should not require a special section of the heap
Cryptographic Library
Secure Memory on Windows

• Uses In-Memory Encryption (Microsoft CryptoAPI)
  • Encryption Key is per-user and generated on boot
  • Encryption Key is stored in nonpaged kernel memory
  • By default, only the VAST Smalltalk process can decrypt

• OpenSSL Dispatcher has been enhanced to
  • Decrypt incoming arguments intended for OpenSSL functions
  • Immediately call the OpenSSL function
  • After the call, re-encrypt the required incoming arguments
Cryptographic Library

Streaming API

- Powerful set of High-Performance OpenSSL Streams
- Two types
  - Source/Sink
    - Socket, File, Memory
  - Filters
    - Digest, Cipher, Base64, Buffer
- Chain them together to create cryptographic pipelines
- Example chain to
  - Perform buffered writes of base64-encoded encrypted data to a file
  - Compute the sha512 hash of the plaint-text

bufferBio | sha512Bio | aes256Bio | base64Bio | fileBio
Cryptographic Library
Message Digests

- Secure one-way hash functions
- Algorithms
  - MD5, RIPEMD160
  - SHA1, SHA2 Family (224, 256, 384, 512)
  - Whirpool
  - Blake2 (OpenSSL 1.1)
- Example:

  ```
  OSSsslDigest sha512
  printableDigest: ‘Hello World’.
  → 958D09788F3C907B1C89A945F478D58C
  ```
Cryptographic Library
Message Authentication Code (MAC)

- Keyed hash function
- Provides both data integrity and authenticity
- Algorithms
  - HMAC
  - CMAC (OpenSSL 1.1)
- Example:
  ```smalltalk
  OpenSSLDigest sha1
  hmacPrintableDigest: 'Hello Smalltalk'
  key: 'secretKey'.
  ```
  → 4510149C9D6216D4460571E16B290312...
Cryptographic Library
Symmetric Ciphers

- Encryption for confidentiality
- Shared secret key
- Block Ciphers
  - AES, Blowfish, Camellia, Cast5, DES, Triple-DES
  - Unauthenticated Modes: CBC, CFB, CTR, OFB, XTS
  - Authenticated Modes: GCM, CCM, OCB
- Stream Ciphers
  - Unauthenticated: ChaCha20
  - Authenticated: ChaCha20-Poly1305
Cryptographic Library
Symmetric Ciphers

- Encrypt Example

"Encrypt"

cipher := OSSslCipher aes_256_ocb.
cData := cipher cipherDataFor: 'Hello Smalltalk'.
authTag := cData tagData.

"Decrypt"

cData := cipher cipherDataFor: cipherText.
cData tagData: authTag.
plainText := cipher decrypt: cData key: key iv: iv
Cryptographic Library
Public/Private Key

- Algorithms using Key Pairs (public and private)
- Use Cases
  - Key Exchange (i.e. agree on a shared key)
  - Non-Interactive Encryption
    - i.e. Encrypted Email
  - Digital Signatures
- Algorithms
  - RSA
  - DSA
  - Diffie-Hellman
Cryptographic Library

Key Derivation

• Derives one or more keys from an initial key material

• Algorithms
  • HKDF
  • PBKDF2
  • Scrypt (OpenSSL 1.1)
Cryptographic Library
Key Derivation

• Password Hashing Example

  “Derive crypto key from a password“
  scrypt := OSSslKDF scrypt keyLength: 16.
  pHash := scrypt derive: ‘password’.

  “Algorithm Params to store with the hash”
  pSalt := scrypt salt.
  pCost := scrypt cost.
  pBlkSz := scrypt blockSize.
  pPara := scrypt parallelization.
  pMaxMem := scrypt maxMemory.
Cryptographic Library
Key Derivation

• Password Hashing Example
  “Verify supplied password with stored hash”
  scrypt := OSSslKDF scrypt
  keyLength: 16
  salt: pSalt
  cost: pCost
  parallelization: pPara
  blockSize: bBlkSz
  maxMemory: bMaxMem.

  (scrypt verify: ‘password’ with: pHash)
  ifTrue: [‘Password is correct’].
SSL/TLS Library

- VA Smalltalk’s existing SSL/TLS support is now built on the new crypto library.
- Inherits the safer memory management features
- More options exposed for SSL/TLS connections
- Gained TLSv1.2 support
- More options for X509 certs
- OpenSSL 1.1 compatible
Thank you for your attention

Questions?