tripwire
CONFIDENCE: SECURED

CppCon 2015

Gwendolyn Hunt
Principal Software Engineer
Tripwire R&D

Secure C++ Programming

Who Am I?

25-years developing software. Mostly developed in C++ since 1994

Current

- **Principal Software Engineer, Tripwire Inc. R&D**
- **Tech lead for new generation security applications**

Previously

- **iMove, Inc. - perpetual startup. Cool tech, didn't make it. C++**
- **IBM – C++ and Java**
- **EDS – Cobol, C and C++**

Why do we care?

- **Almost every day we hear about another data breech**
  - **Target**
  - **Home Depot***
  - **JP Morgan**
  - **Apple**
  - **Google**
  - **White House**
  - **OPM***

Why do we care?

Some Statistics from Online Trust Alliance 2014

- **89% Were Preventable**
- **31% Due to Inside Threats**
- **76% Weak or stolen credentials**
- **29% Via Social Engineering**
- **37% Targeted Attacks (and growing)**
- **12% Lost or stolen devices**

Open Security Foundation and RiskBased Security

- **in 2013 there were 823 million records exposed**

Why do we care?

Types of threats

- **Crime**
- **State-sponsored**
- **Insiders**
- **Script Kiddies**

Breeches often start small

Week or Stolen Credentials or Lost or Stolen Devices

Series of small compromises, eventually leading to the data breech

Target Breech (Krebs on Security - Sept 2015)

- **Once in the network, attackers had full access to all store POS devices**
- **Initial breech was compromised VPN credentials for HVAC company**
  - **to remotely monitor store HVAC systems**
- **BlackPOS malware affected store Card Readers**
  - **updated version of same malware was used in Home Depot Breech**

So what are we going to talk about?

Not going to focus on the large problem

Focus what we as developer's can directly address

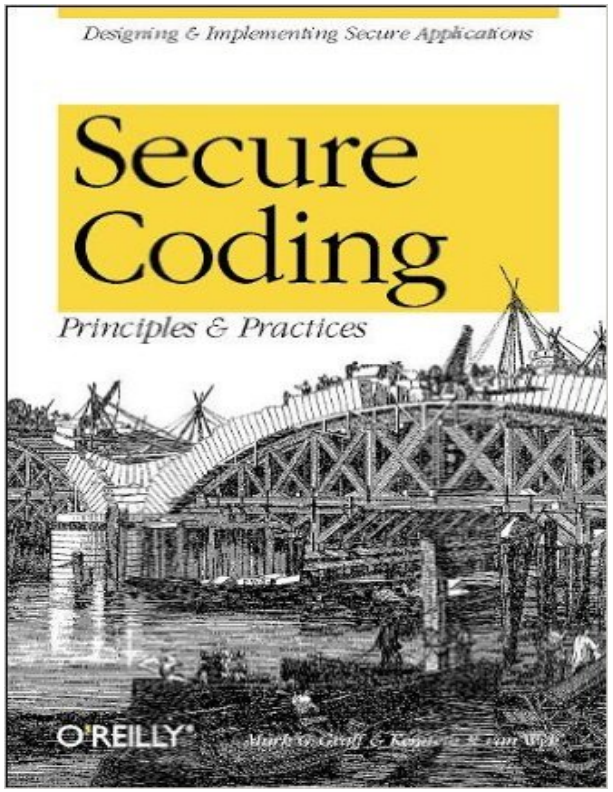
Talk about making our applications more secure

- **Talk about some references**
- **Talk about integer vulnerabilities**
- **Programmer's Toolbox**
- **Secure Programming Processes**
- **Detailed Scenario**

Important References - Books

- **Secure Coding: Principles and Practices**
- **Secure Programming in C and C++, Second Edition**
- **Secure Programming Cookbook for C and C++**

Important References – Secure Coding: Principles and Practices

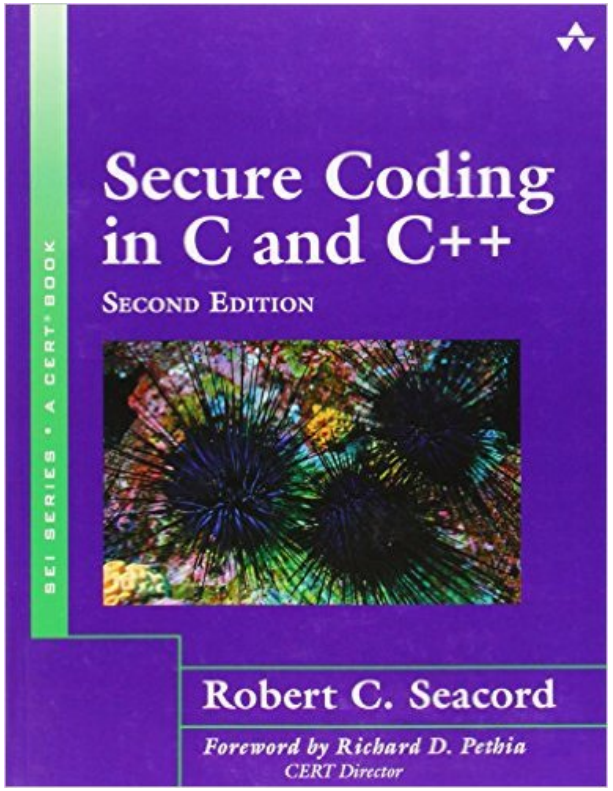


Authors:
Mark G. Graff
Kenneth R. Van Wyk

Publisher: O'Reilly Media;
1st Edition (July 2003)

ISBN-13: 978-0596002428

Important References – Secure Programming in C and C++

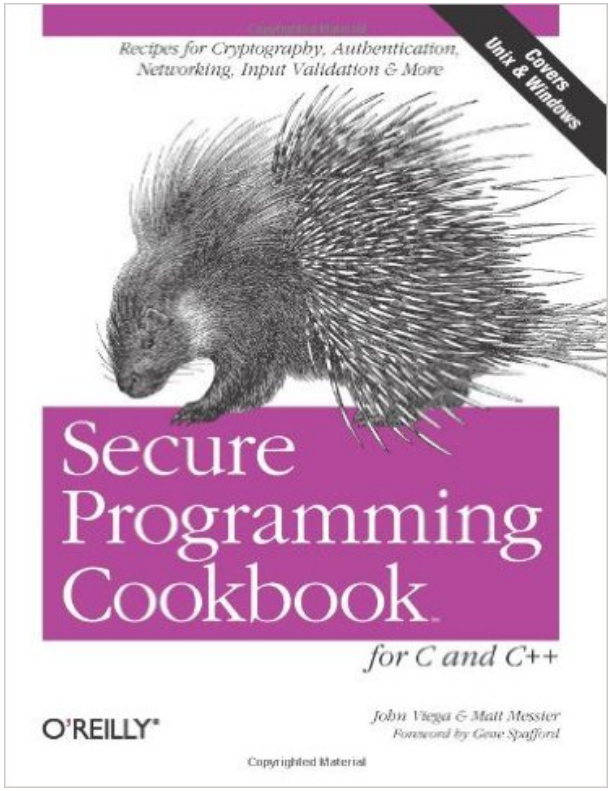


Author: Robert C. Seacord

Publisher: Addison-Wesley Professional; 2nd Edition (April 12, 2013)

ISBN-13: 978-0321822130

Important References – Secure Programming Cookbook for C and C++



Authors:
John Viega
Matt Messier

Publisher: O'Reilly Media;
1st Edition (July 2003)

ISBN-13: 978-0596003944

Important References – Websites

URL: https://www.securecoding.cert.org/confluence/display/seccode/SEI+Cert+Coding+Standards

Common Vulnerabilities

- **Integers**

- **Homework**
  - **Strings**
  - **Arrays**

Common Vulnerabilities - Integers

Unsigned Integer Wrap
Signed Integer Overflow
Conversions

Common Vulnerabilities – Integers

CODE DEMO

unsigned int wrap	examples/ex1
signed int overflow	examples/ex2
conversions	examples/ex3
unbounded arg	examples/ex4 (bad app)
bounded arg	examples/ex5 (better)

Safe Numerics Lib (Boost Incubator)

Common Vulnerabilities – Integers – Unsigned Integer Wrap (Seacord)

Table 5.2. Operator Wrapping

Operator	Wrap	Operator	Wrap	Operator	Wrap	Operator	Wrap
+	Yes	-=	Yes	<<	Yes	<	No
-	Yes	*=	Yes	>>	No	>	No
*	Yes	/=	No	&	No	>=	No
/	No	%=	No	|	No	<=	No
%	No	<<=	Yes	^	No	==	No
++	Yes	>>=	No	~	No	!=	No
--	Yes	&=	No	!	No	&&	No
=	No	|=	No	un +	No	||	No
+=	Yes	^=	No	un -	Yes	?:	No

Common Vulnerabilities – Integers – Signed Integer Overflow (Seacord)

Operator	Overflow	Operator	Overflow	Operator	Overflow	Operator	Overflow
+	Yes	-=	Yes	<<	Yes	<	No
-	Yes	*=	Yes	>>	No	>	No
*	Yes	/=	Yes	&	No	>=	No
/	Yes	%=	Yes	|	No	<=	No
%	Yes	<<=	Yes	^	No	==	No
++	Yes	>>=	No	~	No	!=	No
--	Yes	&=	No	!	No	&&	No
=	No	|=	No	un +	No	||	No
+=	Yes	^=	No	un -	Yes	?:	No

Common Vulnerabilities – Integers – Conversions

Always safe

- **Converting from smaller unsigned ints to larger unsigned ints**

Potentially Not Safe

- **Converting to smaller unsigned int if the resulting value cannot be represented by target value – data lost through truncation**
- **Conversion between signed and unsigned may result in lost data when the target value cannot represented in the new type.**
- **Conversion between unsigned and signed may result in lost data when the target value cannot represented in the new type.**

tripwire

Secure C++ Programmer's Toolbox

Use multiple compilers
Compile and Link Flags that help find defects
Compile and Link Flags that help mitigate attacks

Covered previously in other talks
- **Static Analysis**
- **Clang Sanitizers**

Secure C++ Programmer's Toolbox – Use mulitple compilers

Use a mix - Some compilers are better than others at reporting warnings and errors

- Posix
  - gcc 4.8, 4.4, 4.1
  - Clang 3.7
  - Intel 2015
- OSX
  - Clang (Xcode 7)
- Windows
  - MS VS 2010, VS 2013
  - Intel 2015

Secure C++ Programmer's Toolbox – Flags that help find defects

GCC and Clang Flags

Flag	*Language*	*Comments*
`-Wall -Wextra`	C	Enables many warnings (despite their names, all and extra do not turn on all warnings).
`-Wconversion`	C	Warn for implicit conversions that may alter a value.
`-Wcast-align`	C	Warn for a pointer cast to a type which has a different size, causing an invalid alignment and subsequent bus error on ARM processors.
`-Wformat=2`	C	Increases warnings related to possible security defects, including incorrect format specifiers. -Wformat=2 equiv to -Wformat-nonliteral -Wformat-security -Wformat-y2k
`-fno-common`	C	Prevent global variables being simultaneously defined in different object files.
`-Wstrict-overflow`	C	Warn about optimizations taken due to [undefined] signed integer overflow assumptions. Usually requires minimum optimizations (-O1).
`-Wtrampolines`	C	Warn about trampolines generated for pointers to nested functions. Trampolines require an executable stacks.
`-Woverloaded-virtual`	C++	Warn when a function declaration hides virtual functions from a base class.
`-Wreorder`	C++	Warn when the order of member initializers given in the code does not match the order in which they must be executed.
`-Wsign-promo`	C++	Warn when overload resolution chooses a promotion from unsigned or enumerated type to a signed type, over a conversion to an unsigned type of the same size.
`-fsanitize`	**C++**	**(Clang) address, thread, memory, undefined. Xcode 7 only supports address**

Secure C++ Programmer's Toolbox – Flags that help find defects

MS Visual Studio

Flag	*Language*	*Comments*
`/W4`	C/C++	Warning level 4, which includes most warnings.
`/Wall`	C/C++	Enables all warnings, even those omitted at /W4.
`/analyze`	C/C++	Enables Enterprise Code Analysis.

Secure C++ Programmer's Toolbox – Flags that help mitigate attacks

GCC Additional Flags

Flag	*Comments*
`-DFORTIFY_SOURCE=2`	Uses safer string and memory function when the compiler can deduce the destination buffer sizes.
`-fstack-protector-all`	Stack Smashing Protector (SSP). Improves stack layout and adds a guard to detect stack based buffer overflows.
`-Wcast-align`	Warn for a pointer cast to a type which has a different size, causing an invalid alignment and subsequent bus error on ARM processors.
`-Wl,-z,nodlopen` `-Wl,-z,nodump`	(GCC) Reduces the ability of an attacker to load, manipulate, and dump shared objects.
`-Wl,-z,noexecstack` `-Wl,-z,noexecheap`	(GCC) Data Execution Prevention (DEP). ELF headers are marked with PT_GNU_STACK and PT_GNU_HEAP.
`-Wl,-z,relro`	(GCC) Helps remediate Global Offset Table (GOT) attacks on executables by marking relocations as read-only.
`-Wl,-z,now`	(GCC) Helps remediate Procedure Linkage Table (PLT) attacks on executables.
`-fPIE`	(GCC) Position Independent Executable (ASLR). Used for programs. Both -fPIE (compiler) and -pie (linker) are required.

Windows Additional Flags

Flag	*Comments*
`/GS`	Adds a security cookie (guard or canary) on the stack before the return address buffer stack based for overflow checks. The protection is applied only where heuristic analysis suggests that an overflow is possible.
`/SafeSEH`	Safe structured exception handling to remediate SEH overwrites.
`/NXCOMPAT`	Data Execution Prevention (DEP).
`/dynamicbase`	Address Space Layout Randomization (ASLR).

Secure Programming Processes

- Define and follow a Security Architecture
- Enforce Common Coding Standards
- Use a Code Review System
- Use a Version Control System
- Use a Defect Tracking System
- Use Automated Continuous Integration
- Conduct Fuzz Testing
- Conduct Penetration Testing

Secure Programming Processes – Define and follow a Security Architecture

Define Key Principles and Mechanisms

Examples

Principle: All Internet Domain socket connections are encrypted
Mechanism: TLS 1.2 using high-strength cipher suites and X.509 PKIX compliant with RFC 6125

Principle: Authenticate message source
Mechanism: Use Hash-based Message Authentication codes for all messages utilizing a shared secret.

Principle: All applications must build with no warnings.
Mechanism: Use extra security flags and mitigation flags. Use minimal directed suppression.

Secure Programming Processes - Enforce Common Coding Standards

Controversial - Balance between anarchy and overly restrictive formatting

Helps Readability, Code Reviews and Maintenance

Lot of good examples. Borrow one and modify it for you domain.

Secure Programming Processes – Use a Code Review System

Gets more eyes on the code especially, if required before commit

Example: Reviewboard

Secure Programming Processes – Use a Version Control System

If you're not using version control system your code probably isn't that important.

Needed for teams of developers

Secure Programming Processes – Use a Defect Tracking System

Most effective when integrated with Version Control System and Code Review System

Secure Programming Processes – Use Automated Continuous Integration

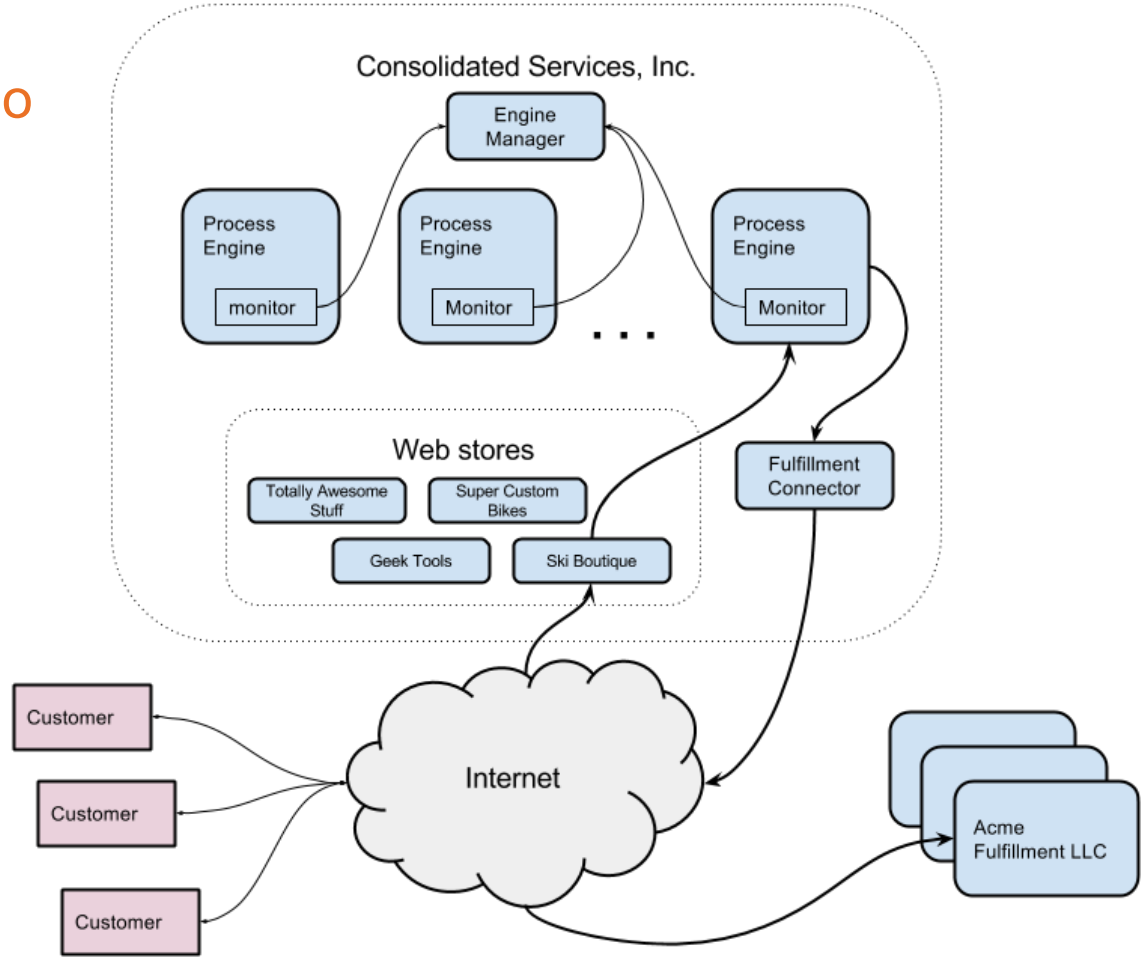
- **Include Static Analysis runs**
- **Include automated fuzzing**
- **Run Daily Integration Tests**
- **Run on all target platform**
- **Make it a big deal if CI breaks from the nightly run**
- **Investigate all failures**

Secure Programming Processes – Conduct Penetration Testing

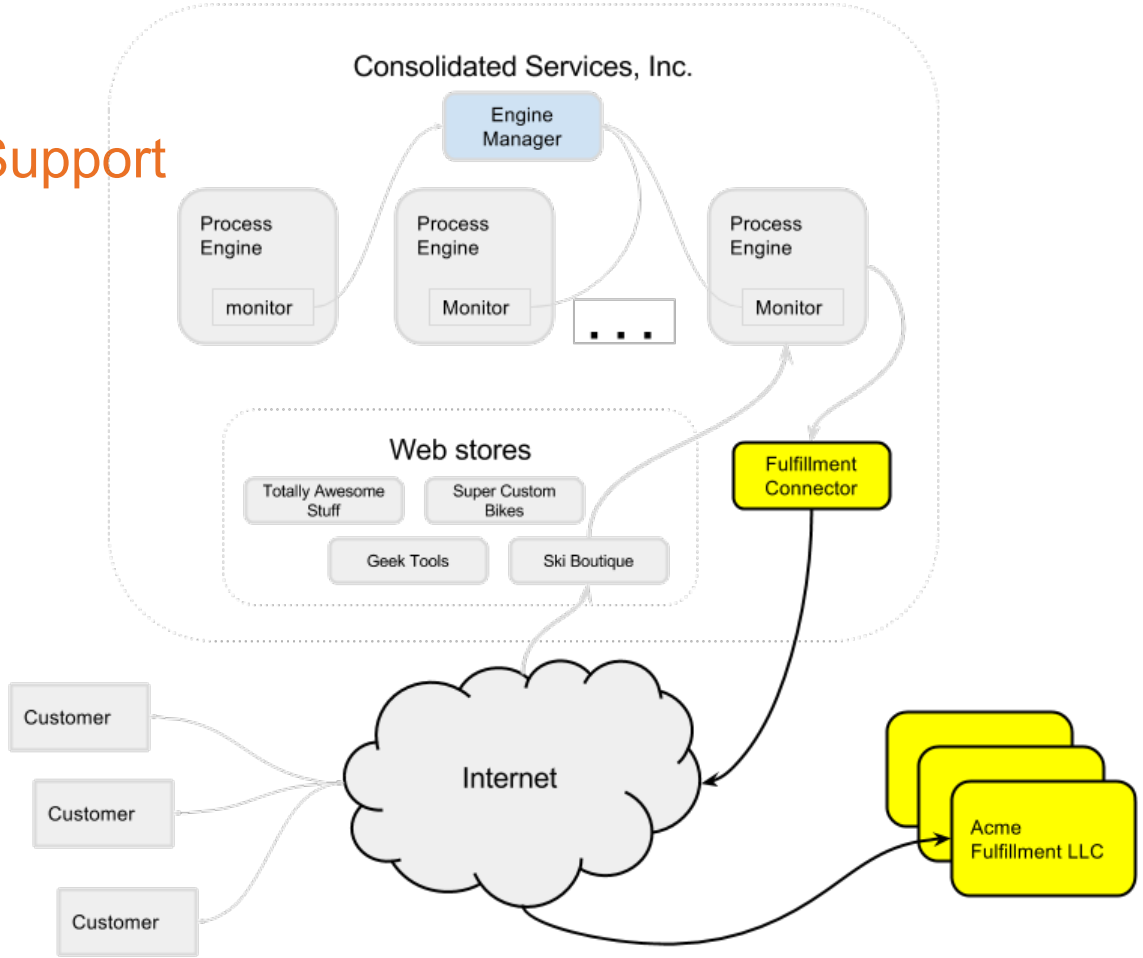
In-house

Thirdparty

Workshop Scenario



Thirdparty Fulfillment Order Support



Workshop Scenarios – Fulfillment Order Support

- **Requirements**
  - **Implement library**
    - **Message handling (parsing and serialization)**
    - **FO Encryption /Decryption**

- **Security Considerations**
  - **Protect the Session**
    - **TLS 1.2 used with high strength ciphers**
      - **ECDHE-RSA-AES128-GCM-SHA256**
    - **Use X.509 Certificates compliant with RFC 6125**
  - **Authenticate the message source**
  - **Encrypt Fulfillment Order**

Fulfillment Library – Protect The Session – TLS 1.2

If you are not using TLS 1.2 you are vulnerable

High Strength Cipher ECDHE-RSA-AES128-GCM-SHA256

- algorithm for key exchange - Eliptical Curve Ephemeral Diffie-Hellman that supports Perfect Forward Secrecy eliminating previous session playback decoding if keys have been compromised.

- algorithm for authentication - RSA uses asymmetric public/private keys where public keys are conveyed in X.509 Identity Certificates and private keys are secured by the owning endpoint.

- algorithm for encrypting the stream contents - AES-GCM an efficient algorithm used for encrypting the data contents. Uses Galois Counter- Method (GCM) instead of Chain Block Coding (CBC)

- algorithm computing cryptographic checksums - SHA256 algorithm used for hashing while message authentication codes that are used to detect accidental or intentional tampering of a block of encrypted data.

Fulfillment Library – Protect The Session – X.509 Certificates

RFC 6125 Identity Verification using X.509 PKIX

Does not rely on the Common Name used in the subject field

Store in subjectAlternateName field

SRV-ID=fo_server._message.fulfillmentco.com
DNS-ID=tp.fulfillmentco.com

SRV-ID=requestor._message.consolidated.com
DNS-ID=s153.consolidated.com

Fulfillment Library – Protect The Session – X.509 Certificates

Key Usage/Extended Key Usage fields

Server Side

EKU serverAuth Must NOT include clientAuth
KU keyEncipherment, keyAgreement (Needed for DHE or ECDHE ciphers), digitalSignature

Client Side

EKU clientAuth Must NOT include serverAuth
KU keyAgreement, digitalSignature

Fulfillment Library – Protect The Session – X.509 Certificate Verification

During cert negotiation after cert path verification:

1. Read subjectAlternateName and parse any service ID. Use the socket connection information about the peer

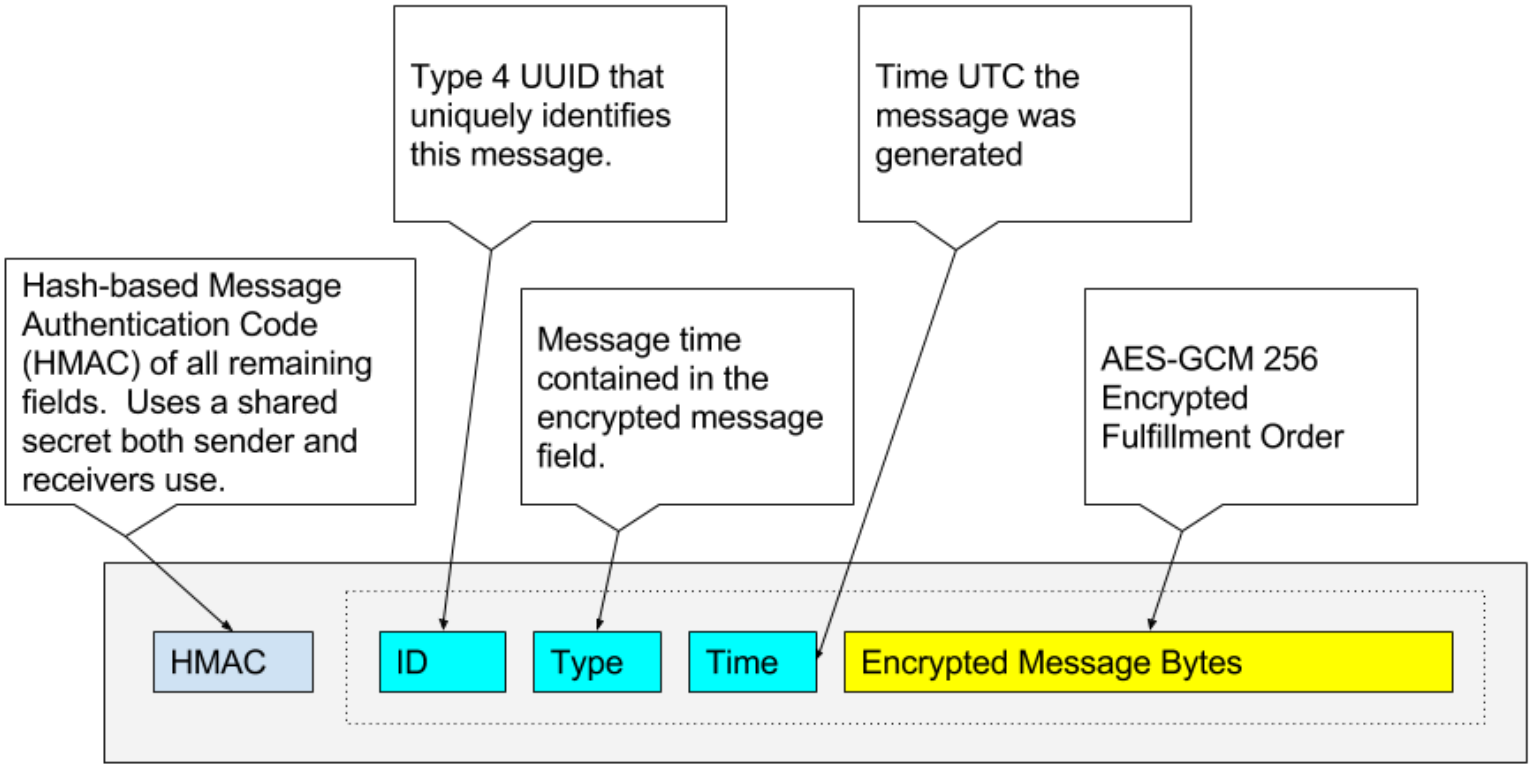
2. The application callback is given the Server ID Certificate, the configured Server's host name and the IP Address of the Bridge when the connection was established prior to TLS Handshaking.

3. At least one SRV-ID and at least one DNS-ID entry must exist in the Server's Certificate. SRV-ID must be unique without wildcards while DNS-ID entries may have leading wildcards.

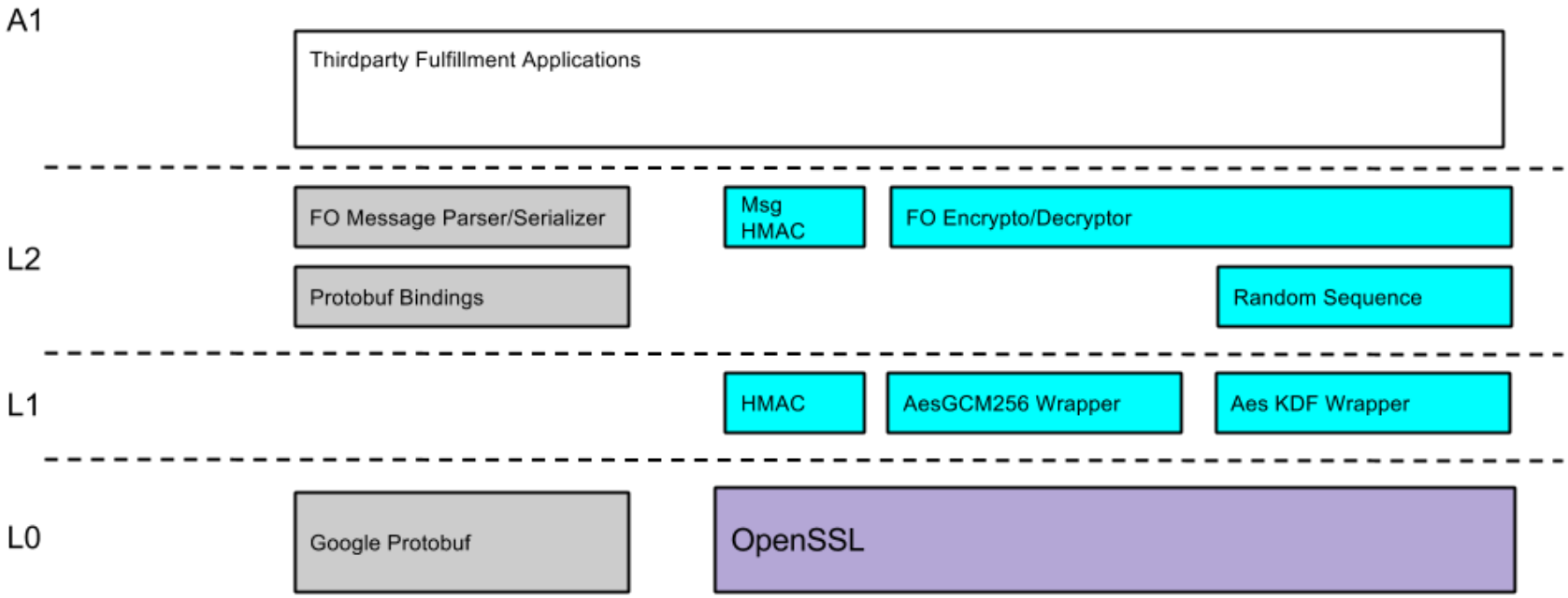
4. The Client constructs a candidate SRV-ID knowing that the service a Server would use **fo_server** with a protocol of **_message** and the hostname passed to the application callback.

5. First the Client compares the candidate SRV-ID to the SRV-ID entries. If there is a match, then the Client does a match of the server name portion of the SRV-ID with the DNS-ID entries. If there is a match, a forward DNS lookup is done on the server name. If the query results contains an IP Address that matches the IP Address reported by the connection, then the Server host name is considered verified.

Fulfillment Library – Authenticate and Encrypt Message



Workshop Scenarios – Fulfillment Library



Fulfillment Library – Encrypted Fulfillment Order – AES GCM 256

Encryption Inputs

- **256-bit symmetric key derived from high quality key derivation function.**
  - ***Key is never sent with the encrypted message***
- **96-bit initialization vector (IV)**
  - ***IV is never used more than once for the same key***
- **plainText**

Encryption Outputs

- **cipher text (same length as plain text)**
- **128-bit TAG generated for tamper detection**

Concatenated into message: TAG (16-bytes) + IV (12-bytes) + cipherText (variable)

Fulfillment Library – Encrypted Fulfillment Order – AES GCM 256

Parse message: TAG (16-bytes) + IV (12-bytes) + cipherText (variable)

Decryption Inputs

- **256-bit symmetric key**
- **96-bit IV**
- **128-bit TAG**
- **cipherText**

Encryption Output

- **plainText**

Fulfillment Library – Encrypted Fulfillment Order – AES GCM 256

CODE DEMO

Thank you!

Example Code: github.com/gmhunt/cppcon15-secp

Gwendolyn Hunt
ghunt@tripwire.com