CES21

California Energy Systems for the 21st Century

California Energy Systems for the 21st Century (CES-21) Program

Secure SCADA Protocol for the 21st Century (SSP-21)

LangSec Workshop at IEEE S&P
24th May, 2018

Adam Crain, Automatak
Prashant Anantharaman, Dartmouth

CES21

California Energy Systems for the 21st Century

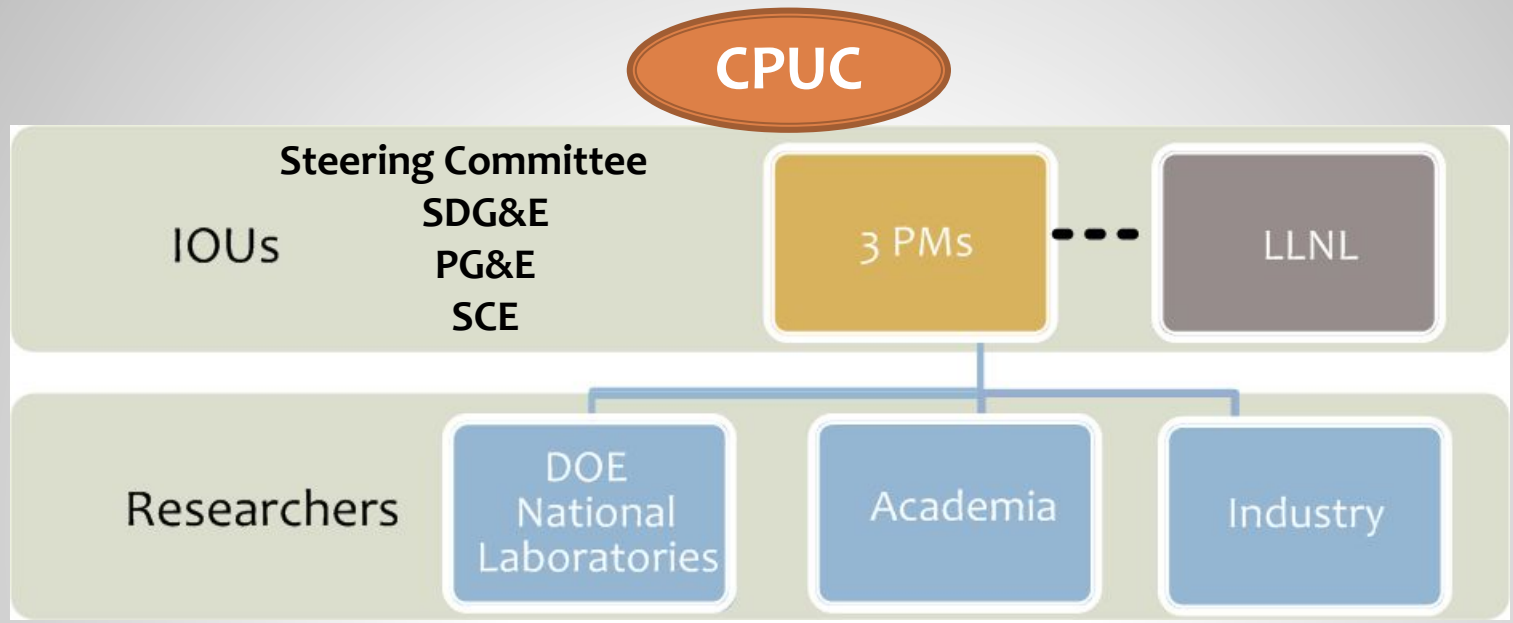
Note on Public Disclosure

The CES-21 Cybersecurity R&D effort is focused on the protection of critical infrastructure, therefore a secure process for reporting and a secure process for deliverables will need to be maintained. Detailed tactics, techniques, and procedures developed for use fall under DHS guidelines and will be marked and handled as

"Protected Critical Infrastructure Information (PCII)"

and not open to the public.

Collaborative Research & Development



The objective of the CES-21 Program is to address challenges of cyber security and grid integration of the 21st century energy system for California through a Collaborative Research and Development Agreement (CRADA). The CES-21 Program utilizes a team of technical experts from Lawrence Livermore National Laboratory (LLNL) and three large Investor-Owned Utilities (IOUs) within the State of California.

Outline

- **Introduction & Review**
- What is SSP21?
- Parsers and Message Formats in SSP21
- Evaluation
- Conclusions and next steps

Back to Langsec 2015 ...

- 2014: 30+ CVEs in DNP3 discovered (Crain / Sistrunk)
- Presentation: “A fuzzing and protocol analysis case study of DNP3”
- Anti-patterns in protocol design and implementation to blame
- “Bolt-On Security Extensions for Industrial Control System Protocols: A Case Study of DNP3 SAv5” [1] (Crain / Bratus)

[1] IEEE Security & Privacy (Volume: 13, Issue: 3, May-June 2015)

It's not all grammar - DNP3

Table 12-4—g3 double-bit binary input static objects

Group	Variation	Subset levels				Request (outstation must parse)		Response (master shall parse)	
		1	2	3	4	Function codes (decimal)	Qualifier codes (hexadecimal)	Function codes (decimal)	Qualifier codes (hexadecimal)
3	0	x	x	x		—	—		
3	0				✓	1 (READ)	00, 01, 06		
3	0				✓	22 (ASSIGN_CLASS)	00, 01, 06		
3	1	x	x	x		—	—	—	—
3	1				✓	1 (READ)	00, 01, 06	129 (RESPONSE)	00, 01
3	2	x	x	x		—	—	—	—
3	2				✓	1 (READ)	00, 01, 06	129 (RESPONSE)	00, 01

Table 14-4—Level 3 implementation (DNP3-L3)

DNP3 OBJECT GROUP & VARIATION			REQUEST Master may issue Outstation shall parse		RESPONSE Master shall parse Outstation may issue	
Group num	Var num	Description	Function codes (dec)	Qualifier codes (hex)	Function codes (dec)	Qualifier codes (hex)
1	0	Binary Input—Any Variation	1 (read) 22 (assign class)	00, 01 (start-stop) 06 (no range, or all)		
1	1	Binary Input—Packed format	1 (read)	00, 01 (start-stop) 06 (no range, or all)	129 (response)	00, 01 (start-stop)
1	2	Binary Input—With flags	1 (read)	00, 01 (start-stop) 06 (no range, or all)	129 (response)	00, 01 (start-stop)
2	0	Binary Input Event—Any Variation	1 (read)	06 (no range, or all) 07, 08 (limited qty)		
2	1	Binary Input Event—Without time	1 (read)	06 (no range, or all) 07, 08 (limited qty)	129 (response) 130 (unsol. resp)	17, 28 (index)
2	2	Binary Input Event—With absolute time	1 (read)	06 (no range, or all) 07, 08 (limited qty)	129 (response) 130 (unsol. resp)	17, 28 (index)
2	3	Binary Input Event—With relative time	1 (read)	06 (no range, or all) 07, 08 (limited qty)	129 (response) 130 (unsol. resp)	17, 28 (index)
10	0	Binary Output—Any Variation	1 (read)	00, 01 (start-stop) 06 (no range, or all)		
10	2	Binary Output—Output status with flags	1 (read)	00, 01 (start-stop) 06 (no range, or all)	129 (response)	00, 01 (start-stop)
12	1	Binary Command—Control relay output block (CROB)	3 (select) 4 (operate) 5 (direct op)	17, 28 (index)	129 (response)	echo of request
			6 (dir. op, no ack)	17, 28 (index)		

GRP	VAR	Type	Description			
0 (0x00)	246 (0xF6)	Attribute	Device Attributes - User-assigned ID code/number			
2 (0x02)	1 (0x01)	Event	Binary Input Event	1 o		
2 (0x02)	2 (0x02)	Event	Binary Input Event - with Absolute Time	7 o		
2 (0x02)	3 (0x03)	Event	Binary Input Event - with Relative Time	3 o		
3 (0x03)	0 (0x00)	Static	Double-bit Binary Input - Any Variations			
3 (0x03)	1 (0x01)	Static	Double-bit Binary Input - Packed Format	2		
3 (0x03)	2 (0x02)	Static	Double-bit Binary Input - Status with Flags	1 o		
4 (0x04)	0 (0x00)	Event	Double-bit Binary Input Event - Any Variations			
4 (0x04)	1 (0x01)	Event	Double-bit Binary Input Event	1 o		
4 (0x04)	2 (0x02)	Event	Double-bit Binary Input Event with Absolute Time	7 o		
4 (0x04)	3 (0x03)	Event	Double-bit Binary Input Event with Relative Time	3 o		
10 (0x0A)	0 (0x00)	Static	Binary Output - Any Variations		1, 22	
10 (0x0A)	1 (0x01)	Static	Binary Output - Packed Format	1 bit	1, 2	129
10 (0x0A)	2 (0x02)	Static	Binary Output - Status with Flags	1 octet	1	129
11 (0x0B)	0 (0x00)	Event	Binary Output Event - Any Variations		1	
11 (0x0B)	1 (0x01)	Event	Binary Output Event - Status	1 octet	1	129,130

A.23.1.2.3 Notes

Read requests and responses shall use qualifier code 0x07 an outstation receives this request, it implicitly indicates current time.

This object can be included in a write request. Write reque value of 1 for this object. When an outstation receives th wants to set the current time in the outstation.

It's not all grammar - GOOSE

Bit	Value	Meaning
0		Leap Second Known
1		ClockFailure
2		Clock not synchronized
3-7		Time accuracy of fractions of second
	00000	0 bit of accuracy
	00001	1 bit of accuracy
	00010	2 bits of accuracy
	00011	3 bits of accuracy
	00100 - 11000	Integer value of number of bits of accuracy
	11001- 11110	Invalid
	11111	unspecified

```
IECGoosePdu ::= SEQUENCE {
        gocbRef              [0]   IMPLICIT VISIBLE-STRING,
        timeAllowedtoLive    [1]   IMPLICIT INTEGER,
        datSet               [2]   IMPLICIT VISIBLE-STRING,
        goID                 [3]   IMPLICIT VISIBLE-STRING OPTIONAL,
        T                    [4]   IMPLICIT UtcTime,
        stNum                [5]   IMPLICIT INTEGER,
        sqNum                [6]   IMPLICIT INTEGER,
        simulation           [7]   IMPLICIT BOOLEAN DEFAULT FALSE,
        confRev              [8]   IMPLICIT INTEGER,
        ndsCom               [9]   IMPLICIT BOOLEAN DEFAULT FALSE,
        numDatSetEntries     [10]  IMPLICIT INTEGER,
        allData              [11]  IMPLICIT SEQUENCE OF Data,
        }

UtcTime ::= OCTETSTRING – format and size defined in 8.1.3.6.

END
```

Bit 0 shall be the leftmost (most significant) bit of the first octet. Bit 7 shall be the rightmost (least significant) bit of the first octet. Bit 8 shall be the leftmost (most significant) bit of the second octet. Bit 15 shall be the rightmost (least significant) bit of the second octet. This shall be continued in that way in further octets.

There are special cases that are individually mapped and do not conform to the general rule. These are the TimeStamp type (specified in 8.1.3.7), quality type (specified in 8.2), TriggerConditions type (specified in 8.1.3.9) and ReasonForInclusion type (specified in 8.1.3.10).

ASN.1 was meant to solve this problem, but falls short

- Heap-based buffer overflows, Denial-of-Service attacks and buffer over-reads are still recurring.
- Crypto++ and OpenSSL also had issues with ASN.1 parsing.

CVE-2018-0739	Constructed ASN.1 types with a recursive definition (such as can be found in PKCS7) could eventually exceed the stack given malicious input with excessive recursion. This could result in a Denial Of Service attack. There are no such structures used within SSL/TLS that come from untrusted sources so this is considered safe. Fixed in OpenSSL 1.1.0h (Affected 1.1.0-1.1.0g). Fixed in OpenSSL 1.0.2o (Affected 1.0.2b-1.0.2n).
CVE-2017-9023	The ASN.1 parser in strongSwan before 5.5.3 improperly handles CHOICE types when the x509 plugin is enabled, which allows remote attackers to cause a denial of service (infinite loop) via a crafted certificate.
CVE-2017-11496	Stack buffer overflow in hasplms in Gemalto ACC (Admin Control Center), all versions ranging from HASP SRM 2.10 to Sentinel LDK 7.50, allows remote attackers to execute arbitrary code via malformed ASN.1 streams in V2C and similar input files.
CVE-2017-1000416	axTLS version 1.5.3 has a coding error in the ASN.1 parser resulting in the year (19)50 of UTCTime being misinterpreted as 2050.
CVE-2016-9939	Crypto++ (aka cryptopp and libcrypto++) 5.6.4 contained a bug in its ASN.1 BER decoding routine. The library will allocate a memory block based on the length field of the ASN.1 object. If there is not enough content octets in the ASN.1 object, then the function will fail and the memory block will be zeroed even if its unused. There is a noticeable delay during the wipe for a large allocation.
CVE-2016-9132	In Botan 1.8.0 through 1.11.33, when decoding BER data an integer overflow could occur, which would cause an incorrect length field to be computed. Some API callers may use the returned (incorrect and attacker controlled) length field in a way which later causes memory corruption or other failure.
CVE-2016-7053	In OpenSSL 1.1.0 before 1.1.0c, applications parsing invalid CMS structures can crash with a NULL pointer dereference. This is caused by a bug in the handling of the ASN.1 CHOICE type in OpenSSL 1.1.0 which can result in a NULL value being passed to the structure callback if an attempt is made to free certain invalid encodings. Only CHOICE structures using a callback which do not handle NULL value are affected.
CVE-2016-6891	MatrixSSL before 3.8.6 allows remote attackers to cause a denial of service (out-of-bounds read) via a crafted ASN.1 Bit Field primitive in an X.509 certificate.
CVE-2016-6129	The rsa_verify_hash_ex function in rsa_verify_hash.c in LibTomCrypt, as used in OP-TEE before 2.2.0, does not validate that the message length is equal to the ASN.1 encoded data length, which makes it easier for remote attackers to forge RSA signatures or public certificates by leveraging a Bleichenbacher signature forgery attack.
CVE-2016-5080	Integer overflow in the rtxMemHeapAlloc function in asn1rt_a.lib in Objective Systems ASN1C for C/C++ before 7.0.2 allows context-dependent attackers to execute arbitrary code or cause a denial of service (heap-based buffer overflow), on a system running an application compiled by ASN1C, via crafted ASN.1 data.
CVE-2016-4421	epan/dissectors/packet-ber.c in the ASN.1 BER dissector in Wireshark 1.12.x before 1.12.10 and 2.x before 2.0.2 allows remote attackers to cause a denial of service (deep recursion, stack consumption, and application crash) via a packet that specifies deeply nested data.
CVE-2016-4418	epan/dissectors/packet-ber.c in the ASN.1 BER dissector in Wireshark 1.12.x before 1.12.10 and 2.x before 2.0.2 allows remote attackers to cause a denial of service (buffer over-read and application crash) via a crafted packet that triggers an empty set.
CVE-2016-2842	The doapr_outch function in crypto/bio/b_print.c in OpenSSL 1.0.1 before 1.0.1s and 1.0.2 before 1.0.2g does not verify that a certain memory allocation succeeds, which allows remote attackers to cause a denial of service (out-of-bounds write or memory consumption) or possibly have unspecified other impact via a long string, as demonstrated by a large amount of ASN.1 data, a different vulnerability than CVE-2016-0799.
CVE-2016-2522	The dissect_ber_constrained_bitstring function in epan/dissectors/packet-ber.c in the ASN.1 BER dissector in Wireshark 2.0.x before 2.0.2 does not verify that a certain length is nonzero, which allows remote attackers to cause a denial of service (out-of-bounds read and application crash) via a crafted packet.

Source: cve.mitre.org

Outline

- Introduction & Review
- **What is SSP21?**
- Parsers and Message Formats in SSP21
- Evaluation
- Conclusions and next steps

Secure SCADA Protocol for the 21st Century (SSP-21)

Application security:

- shared secrets
- one-time shared secrets (QKD)
- pre-shared public keys
- certificate chains

Legal review for public release nearing completion

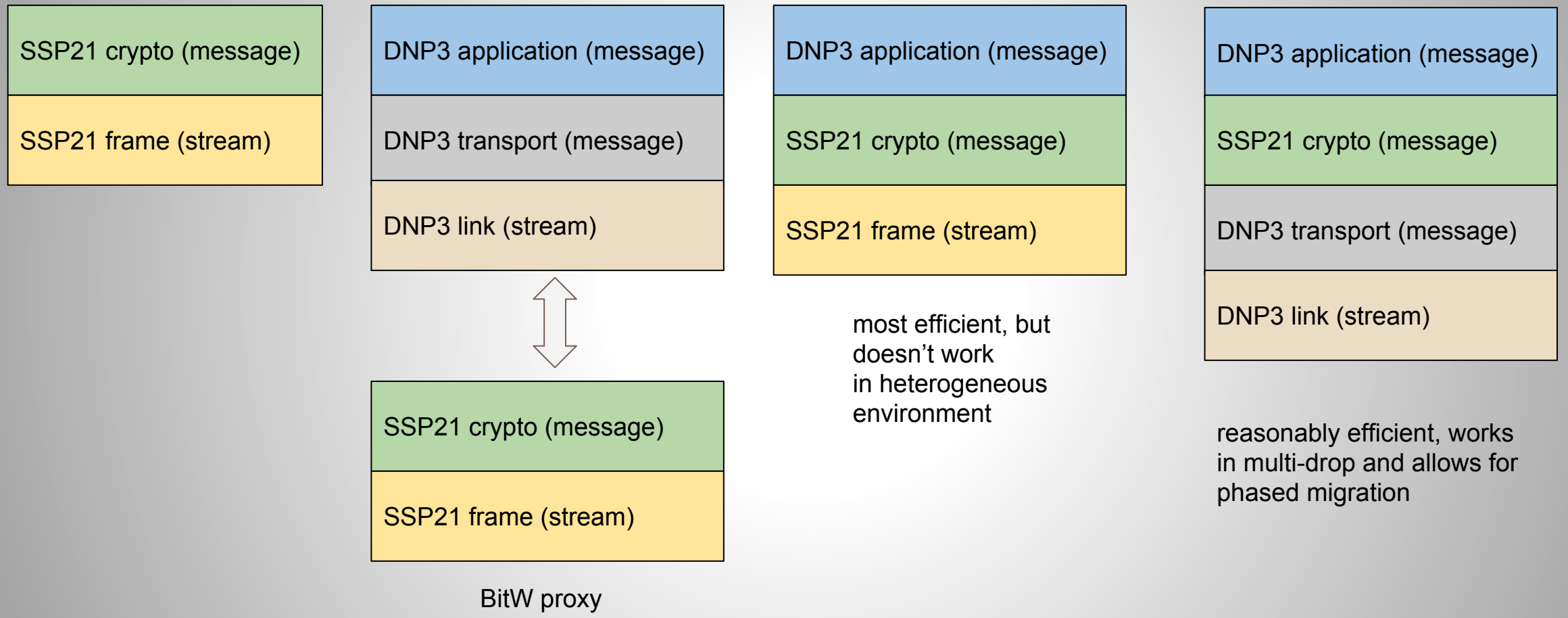
Why not TLS?

- **Many bells and whistles**
  - Easier to misconfigure
  - Creates extra attack surface
- **PKI based on x.509**
  - Hotbed for security issues
  - Irrelevant metadata for ICS
- **TLS 1.3**
  - No authentication-only cipher suites
  - PFS-only! No passive monitoring

"Bugs are not randomly distributed; certain flaming hoops are reliably problematic" – Dan Kaminsky

https://www.ioactive.com/pdfs/PKILayerCake.pdf

Example: Apply SSP21 for DNP3



Outline

- Introduction & Review
- What is SSP21?
- **Parsers and Message Formats in SSP21**
- Evaluation
- Conclusions and next steps

SSP21 Syntax

- Defines and specifies how the message is serialized.
- Messages are special *structs* and have a constant first field of a *function* enumeration.

```
struct <struct-name> {
  <field1-name> : <field1-type>
  <field2-name> : <field2-type>
  ...
  <field3-name> : <field3-type>
}
```

```
message <message-name> {
  function : enum::Function::<function-name>
  <field1-name> : <field1-type>
  <field2-name> : <field2-type>
  ...
  <field3-name> : <field3-type>
}
```

```
enum <enum-name> {
  <name1> : <value1>
  <name2> : <value2>
  ...
  <nameN> : <valueN>
}
```

Handshake Request Message Format

message **BeginHandshakeRequest** {

function	: enum::Function:: BEGIN_ HANDSHAKE_REQUEST
version	: U16
handshake_mode	: enum::HandshakeMode
crypto_spec	: struct::CryptoSpec
constraints	: struct::Constraints
ephemeral_data	: SeqOf[U8]
mode_data	: SeqOf[U8]

}

- Shared secret
- Pre-shared public key
- Certificates

Algorithms (no negotiation!)

Limits on time / nonce (PLP)

Interpreted based on handshake mode

message **BeginHandshakeReply** {
  function : enum::Function:: BEGIN_ HANDSHAKE_REPLY
  ephemeral_data: SeqOf[U8]
  mode_data: SeqOf[U8]
}

Session Message Format

```
message SessionData {
  function    : enum::Function::SESSION_DATA
  metadata    : struct::AuthMetadata
  user_data   : SeqOf[U8]
  auth_tag    : SeqOf[U8]
}
```

clear-text or encrypted

truncated MAC or AEAD Tag

```
struct AuthMetadata {
   nonce : U16
   valid_until_ms : U32
}
```

always clear-text, but covered by authentication tag

TTL since “session start”

Certificates also defined in grammar

```
message CertificateEnvelope {
    certificate_data    :   SeqOf[U8]
    algorithm           :   enum::SIGNATURE_ALGORITHM
    signature           :   SeqOf[U8]
}

message CertificateBody {
    serial_number       :   U64
    valid_after         :   U64
    valid_before        :   U64
    signing_level       :   U8(max = 6)
    public_key_type     :   enum::PublicKeyType
    public_key          :   SeqOf[U8]
    extensions          :   SeqOf[struct::ExtensionEnvelope](max = 5)
}
```

Ed25519

- 32-byte 25519 keys
- 64-byte 25519 signatures
- No extensions
- **133 bytes!**

```
object BeginHandshakeRequest extends Message {

  override def name: String = "BeginHandshakeRequest"

  def function = CryptoFunction.requestHandshakeBegin

  override def fields: List[Field] = List(
    U16("version"),
    Enum(HandshakeMode),
    StructField("spec", CryptoSpec),
    StructField("constraints", SessionConstraints),
    SeqOfByte("ephemeral_data"),
    SeqOfByte("mode_data")
  )

}
```

Message DSL and code generator

```cpp
struct BeginHandshakeRequest final : public IMessage
{
    BeginHandshakeRequest();

    BeginHandshakeRequest(
        uint16_t version,
        HandshakeMode handshake_mode,
        const CryptoSpec& spec,
        const SessionConstraints& constraints,
        const seq32_t& ephemeral_data,
        const seq32_t& mode_data
    );

    virtual ParseError read(seq32_t input) override;
    virtual FormatResult write(wseq32_t& output) const override;
    virtual void print(IMessagePrinter& printer) const override;

    static const Function function = Function::request_handshake_begin;

    IntegerField<openpal::UInt16> version;
    EnumField<HandshakeModeSpec> handshake_mode;
    CryptoSpec spec;
    SessionConstraints constraints;
    SeqByteField ephemeral_data;
    SeqByteField mode_data;

};
```

Generates C++ headers and implementation.

Langsexy properties of messages

- Bounded size types => no heap allocation
  - only machine integers up to U64
  - limits in grammar for depth of certificate chain
  - no recursive types
- No “choice” aka polymorphism
  - always leads to loss of type safety and dynamic casting
- No optional fields
  - Don’t these always lead to null ptr dereference?
- No string types (yet)
  - May have to relax this as cert format finalizes?

Outline

- Introduction & Review
- What is SSP21?
- Parsers and Message Formats in SSP21
- **Evaluation**
- Conclusions and next steps

Fuzzing w/ AFL

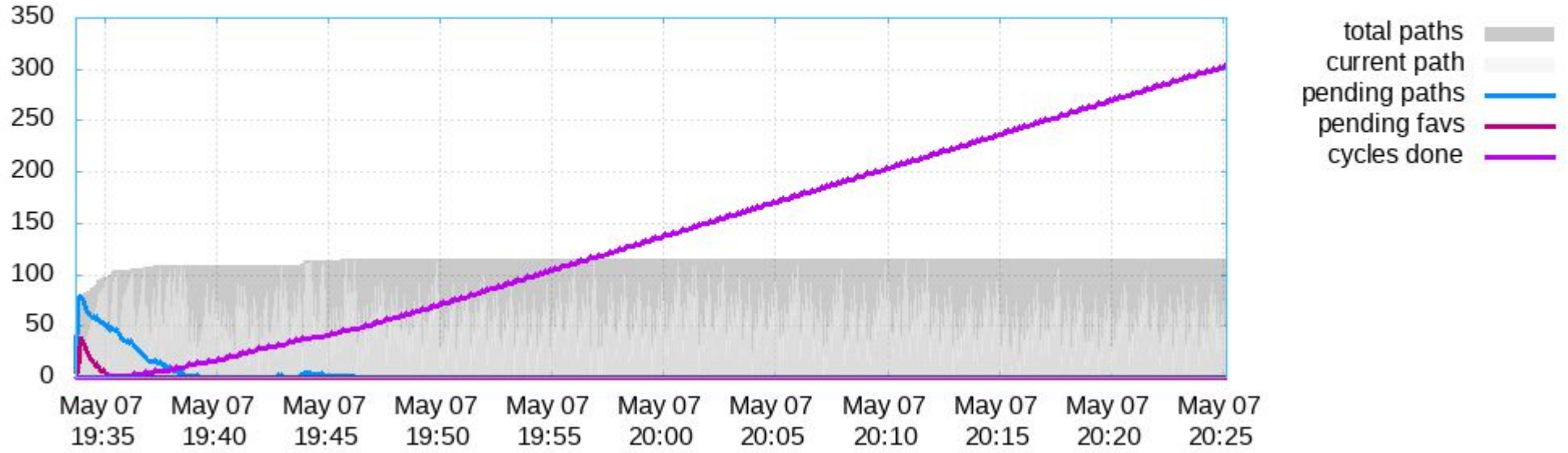
1. Create test harness that reads stdin
   a. pass input to each message parser
   b. if no error, print message to stdout
   c. option to output valid seed for each message

2. Compile w/ instrumentation, run until no new paths

3. Verify coverage using afl-cov (gcov based)

No new paths after only ~20 minutes

Banner: ssp21-afl
Directory: findings/
Generated on: Mon May 7 13:25:13 PDT 2018



Excellent coverage of parsing primitives and composition

```
37        41 :         ParseError read(seq32_t& input)
38           :         {
39        41 :             this->clear();
40           :
41           :             uint32_t count;
42        41 :             auto cerr = VLength::read(count, input);
43        41 :             if (any(cerr)) return cerr;
44           :
45       192 :             while (count > 0)
46           :             {
47        90 :                 StructType item;                ;
48        90 :                 auto serr = item.read(input);
49       103 :                 if (any(serr)) return serr;
50           :
51        78 :                 if (!this->push(item))
52           :                 {
53         1 :                     return ParseError::impl_capacity_limit;
54           :                 }
55           :
56        77 :                 --count;
57           :             }
58           :
59        25 :             return ParseError::ok;
60           :         }
```

```
           :     class MessageParser : private openpal::StaticOnly
           :     {
           :
           :     public:
           :
           :         /// Enforces that the first byte is the expected function and expects all data to be consumed.
           :         template <typename ReadFields>
       345 :         static ParseError read_message(const seq32_t& input, Function expected, const ReadFields& read_fields)
           :         {
       345 :             seq32_t copy(input);
           :
       345 :             EnumField<FunctionSpec> func;
       345 :             auto err = func.read(copy);
       345 :             if (any(err)) return err;
       249 :             if (func != expected) return ParseError::unexpected_function;
           :
        80 :             err = read_fields(copy);
        80 :             if (any(err)) return err;
           :
           :             // top level messages must always fully read the input
        22 :             return copy.is_empty() ? ParseError::ok : ParseError::too_many_bytes;
           :         }
           :
           :         template <typename T, typename... Args>
      1334 :         static ParseError read_fields(seq32_t& input, T& value, Args& ... args)
           :         {
      1334 :             auto err = value.read(input);
      1334 :             if (any(err)) return err;
      1108 :             return read_fields(input, args...);
           :         }
           :
           :     private:
           :
       256 :         static ParseError read_fields(seq32_t&)
           :         {
       256 :             return ParseError::ok;
           :         }
           :     };
```

Outline

- Introduction & Review
- What is SSP21?
- Parsers and Message Formats in SSP21
- Evaluation
- **Conclusions and next steps**

SSP-21 progress and updates

- Phases 1 -> 7 (completed)
  - Specification w/ pre-shared public keys
  - Reference implementation with pre-shared public keys
  - Lab testing of serial BitW with pre-shared public keys
  - Extend specification with certificate support
  - Extend reference implementation and BitW with certificate support
  - Laboratory integration with Quantum Key Distribution (QKD)
- DOE Cybersecurity for Energy Delivery Systems (CEDs)
  - Evaluation of protocol and “Industrial Key Infrastructure”

What does SSP21 mean for LangSec?

- Machine readable spec format, that doesn't contain the ambiguity of ASN.1.
  - With its imminent widespread adoption in the energy sector, we could be expunging a large number of input-handling vulnerabilities in the underlying SCADA protocols.
- The success story of SSP21 with its easy to read spec, and a simple code generator must serve as a success story for the rest of the industry to follow.
- Well-factored parsers are more maintainable and extensible.
- Extending the message format to include strings.

Thank You

Questions?

Adam Crain: jadamcrain@automatak.com
Prashant : pa@cs.dartmouth.edu

SSP-21 Open Source Project

Companies, Researchers, Developers - Participation Welcome!

slack

Contact: rcorrigan@qubitekk.com

GitHub

Contact: jadamcrain@automatak.com