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
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.
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)

It’s not all grammar - DNP3

A.23.1.2.3 Notes

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

This object can be included in a write request. Write request value of 1 for this object. When an outage receives it, it wants to set the current time in the outage.
It’s not all grammar - GOOSE

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

IECGoosePdu ::= SEQUENCE {
    gcobRef [0] IMPLICIT VISIBLE-STRING, 
    timeAllowedToLive [1] IMPLICIT INTEGER, 
    dataSet [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, 
    numDataSetEntries [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.

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

BitW proxy

SSP21 crypto (message)

SSP21 frame (stream)

DNP3 application (message)

DNP3 transport (message)

DNP3 link (stream)

SSP21 crypto (message)

SSP21 frame (stream)

DNP3 application (message)

DNP3 transport (message)

DNP3 link (stream)

most efficient, but doesn’t work in heterogeneous environment

reasonably efficient, works in multi-drop and allows for phased migration
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.

```c
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]
}

message BeginHandshakeReply {
  function : enum::Function::BEGIN_HANDSHAKE_REPLY
  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
Session Message Format

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

struct AuthMetadata {
  nonce : U16
  valid_until_ms : U32
}

- clear-text or encrypted
- truncated MAC or AEAD Tag
- always clear-text, but covered by authentication tag
- TTL since “session start”
Certificates also defined in grammar

```plaintext
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")
    )
}
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
Excellent coverage of parsing primitives and composition
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