**OpenC2 Language Security – Initial Draft**

**Introduction**

* Emphasis placed on separation of problem space and solution space
* Focus of OpenC2 is the language, not the ACD system
* Introduce concept that a poorly designed/implemented OpenC2 may enrich an environment for potential adversaries
* Objective of OpenC2 should be to provide rapid response while not introducing new avenues of exploitation

**Threat Landscape**

* Control planes of all systems/networks is a primary target (think keys to the kingdom) since control planes are often the enablers of access.
* CND systems are also key targets as part of subverting access controls and detection.
* Many attacks focus on protocol/language exploitation based on lax enforcement and input validation.

**Operating Environment**

* Mixed environments.
* Mixed device smarts.
* Inheritance of legacy devices and architectures.
* Varied regulatory and compliance landscape leads to varied design choices.
* The OpenC2 language should be thin and flexible enough to fit within a wide range of environments.

**Security Implications of Language Structure**

* Only core security elements should be added to syntax
  + applicable to all targets and actuators
  + applicable to all contexts
* Context-specific security elements as specifiers
  + ignored by devices that don't have the required capability
  + not mandatory in all domains
* Security analysis must account for possibility of specifier being ignored
* Enduring, enclave-wide context-specific security requirements can be articulated as policy, rather than sent with each command
* Hooks for future development
* Thoughtful distinction between the payload syntax and the message wrapper

**Security Topics**

**Non-Repudiation**

The current OpenC2 syntax only supports sending and receiving messages but does not have any mechanisms to ensure the transferred message has been sent and received by the parties claiming to have sent and received the message. Nonrepudiation is a way to guarantee that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message.

**Integrity / Man in the Middle**

Currently there is no message protection when using the OpenC2 syntax. When a message is received by an OpenC2 target, it should ask two questions: whether I trust the sender (another target or orchestrator) and whether it created this message. Assuming that the sender trust has been established one way or another, the target has to be assured that the message it is looking at was indeed issued by the server, and not altered along the way

**Confidentiality**

Often times, it is not sufficient to ensure the integrity of an OpenC2 message; in many cases it is also desirable that nobody can see the message data that is passed around and/or stored locally. It may apply to the entire message being processed, or only to certain parts of it. In either case, some type of encryption is required to conceal the content. Normally, symmetric encryption algorithms are used to encrypt bulk data, since it is significantly faster than the asymmetric ones. Asymmetric encryption is then applied to protect the symmetric session keys, which, in many implementations, are valid for one communication only and are subsequently discarded.

Applying encryption requires conducting an extensive setup work, since the communicating parties now have to be aware of which keys they can trust, deal with certificate and key validation, and know which keys should be used for communication.

**Authentication**

OpenC2 syntax does support multiple users. However relying on the syntax alone does not provide actual user authentication services and this is an external dependency.

Even between system components; orchestrator to target, OpenC2 does not provide system authentication. There are modifiers to support user id and even related permissions; however, an id is not an appropriate method for user authentication. Consider the following construct. Id=jsmith isn’t enough to use the language command construct to support full authentication.

ALLOW (

[target (type=user,[ id=jsmith, priv=read, edit])],

[actuator (type=network.access-control,[ id=cont23425]) ],

[period = 30 days]

)

**Authorization**

There are actions within the OpenC2 language that can be grouped by their general activity. Each group of actions may need to have some level of user authentication to allow such actions to be performed. One set of actions that control permissions and accesses are a desired area of interest of an attacker that can use these actions for his advantages. The OpenC2 commands are DENY, CONTAIN, ALLOW, which either of these used in the wrong context would lead to network compromise. Based on the security issues with authentication, the same concerns are with command authorization. OpenC2 syntax does not have inherited user schemas or even user to command mapped permissions that allow or not allow the ACTION to be performed.

Consider the following:

STOP (

[target (type=process, [id=123345])],

[actuator (type=endpoint ,[id=9876t6])],

[immediate]

)

The STOP construct would stop a process on an endpoint with an immediate action when the message is received. If this was a mission critical device that received the command without authorization, the network could be rendered inoperable.

**Accounting**

The accounting leg of AAA services is virtually indistinguishable from auditing. Accounting services typically capture the who, when, and what aspects of access to systems and the commands executed on those systems.

**Auditing**

One of the largest challenges of using OpenC2 is the ability to audit the activity that comes from the lightweight and often sparse use of the OpenC2 messaging framework to get the fastest performance with little to no oversight.  In a real world implementation of OpenC2, providing automated actions and changes will require a digital fingerprint on the activity and results that were accomplished via the OpenC2 framework and implementation.

Actions and the following results that are the direct result of OpenC2 should be recorded and analyzed for security areas such as forensics, secure implementation, security architecture of impact changes within the environment, and completion of such tasks.  Currently there are no enforcements within the OpenC2 framework that would directly accomplish such actions.

**Replay**

Even a valid message may present a danger if it is utilized in a "replay attack". i.e. it is sent multiple times to the server to make it repeat the requested operation. This may be achieved by capturing an entire message, even if it is sufficiently protected against tampering, since it is the message itself that is used for attack now. Usual means to protect against replayed messages is either using unique identifiers (nonces) on messages and keeping track of processed ones, or using a relatively short validity time window.

**Verb Whitelisting**

Verb Whitelisting is a huge area of concern when it comes to OpenC2. Many devices such as a firewall are capable to accept commands such as DELETE, MODIFY, ACCEPT, DENY, START, STOP, RESTART however even if such a device is capable of accepting the commands and act upon an action, not all actions should take place especially during production operating hours. Actuators and Target devices will blindly act upon actions, in essence OpenC2 command hierarchy is too much of an enabling system without certain capability checks in place.

**Security Impacts on Performance**

Resources required to process additional layers of security (CPU, RAM, time, etc…)

Key management introduces additional complexity

Vendor integration challenges (Example: Subject Alternative Name parsing varies by vendor)

Modifications to the OpenC2 language structure in support of complex tasks such as encryption should be kept to a minimum to avoid re-inventing already established standards and taxing devices which might not be capable of the required processing overhead. This is an area where the deployed environment can best leverage existing and compatible capabilities to perform such services.

**Architecture / Implementation Considerations**

*Peer-to-Peer*

Peer-to-peer architectures can present many atypical issues not found on other domain-based systems. Secure connections have different KMI challenges and basic issue of trust must be addressed differently.

*(Multi) Hub and Spoke*

*Strict Type Enforcement*

Strict type enforcement and the related input validation may prove challenging with the current loose structure (everything is optional).

*Integration with Configuration Management*

The tracking (auditing) and reporting of changes initiated by the OpenC2 ecosystem needs to feed back to the CM solution as a matter of both best practice and regulatory and certification & accreditation compliance.

*Out of Band Management (OOBM)*

OOBM is a common best practice with renewed focus based on the threat landscape listed earlier in this document. C2 systems are prime objectives for bad actors and OOBM offers another layer in the defense-in-depth model.

*Domain Federation*

A federation trust model is often found in partner organizations with explicit trust and a shared security and/or accreditation boundary.

*Certification and Accreditation (C&A) – Regulatory Compliance*

Many environments will face a range of regulatory compliance or certification and accreditation requirements. A combination of the native OpenC2 language feature set and deployed architecture must address fundamental security features (auditing, CIA, AAA, encryption) in order to achieve compliance or Authority to Operate (ATO).