OpenC2

Abstract: Cyber attacks operate at machine speed. Effective response to attacks must operate at machine speed. Security solutions must not only be faster and smarter, they must be able to work together to create a concerted automatic cyber defense. Network defenses such as firewalls, intrusion detectors, encryption devices, sand boxes, and honeypots must be able not only to share information, but also to orchestrate their actions, and the actions of a variety of network and host control devices, to dynamically adjust their operating conditions to counter act adversarial activity. An open standards approach to translating the operational and defensive intent of an organization into orchestrated and automated system control actions is needed.

G2 Inc. is a Maryland based cybersecurity innovation center. We provide software, systems, and security engineering services to our nation's critical agencies and infrastructure. G2 has been delivering quality cybersecurity solutions and services to its clients for the last 13 years in security automation, analytics, and high speed processing

Company: G2 Inc.

Authors:

Pat Muoio, Director of R&D, Paul Green, CTO

OpenC2.org

The Proposal

We propose to initiate an effort to develop an open standards driven orchestration language based on the nouns and verbs required to encode human intent and decisions as machine-readable instructions to enable automated courses of action.

It is key that this language be compact yet expressive of a large majority of the potential actions required in a general way. It would seem that while many of the nouns required to support this effort have already been effectively defined either practically or formally (e.g. IP address, URLs and filenames) there is a lack of definition of the verbs required to express the courses of action.

We propose the creation of a core set of verbs as well as an extensible course of action message format. Building on these root verbs, we can expand the expressiveness of the language in a context-specific way by adding adverbs and objects to further specify the general activities. This strategy allows us to achieve broad interoperability while only standardizing and maintaining a small set of verbs. The extensions will be created by users and vendors and need not be universal to be useful in context.

The core language can be implemented in a variety of systems to perform the secure delivery and management of courses of action messages, again in a context-specific way. (e.g. OpenC2 could be potentially transported within AMQP). These systems can vary in assurance level and complexity. They can address issues like identity management and roots of trust in a tailored way. A reference implementation of an orchestration system will prove the effectiveness of this approach.

The Plan

Develop an orchestration language

The first step on the path to OpenC2 is the development of the core of an orchestration language to denote the courses of action to be directed by the orchestrator. We must define a parsimonious, but extensible, set of verbs to express things to be done. We must also identify, and perhaps augment, relevant existing standards (such as STIX, TAXI, CybOX, SCAP, SNMP) that define a relevant set of nouns to express system state and action triggers.

Future activities could define a syntax and logic to provide a standard way of orchestrating complex courses of action that can be used as a pattern for specific system implementations.

Two important design criteria are central to the success of this effort: practice strict discipline in establishing the core of this language, and leverage existing work wherever possible.

The language will be declarative, not prescriptive. It will say what needs to be done but not specify how it should be done. The device will interpret the general command in the context of its capabilities and its environment. The core set of verbs for this language must be small, and expressive of a broad set of general actions. Context-specific adverbs will provide extensibility while enabling the set of verbs to remain tightly constrained. This core discipline will enable us to keep the language lightweight by contextualizing special concerns and will thereby enable flexibility.

This language is not being written on a blank slate, and it makes sense to leverage existing standards wherever possible. OpenC2 will take advantage of appropriate head starts and add functionality that enables devices in these already-specified domains to interact with devices that are not similarly specified. The existing standards usefully express the "how" to address the OpenC2 "what."

Design an example execution platform

Next we will design an execution platform that takes as input the threats, indicators, and statements of machine state, decides on the correct course of action, and deploys that course of action over the system components.

This platform will have several key components: a scout that stands on lookout for threats and anomalous system measures expressed in the orchestration language, an executive that decides whether any individual or combination of these conditions is a trigger for a course of action, and a dispatcher that delivers the executive's decision to the system components (expressed in the orchestration language) and, optionally, interpreters that translate machine metrics and system alerts into the orchestration language.

Create a reference implementation

To aid in the adoption of this open source language and platform, we will create a reference implementation which will demonstrate how a complex system response can be coordinated over a number of monitoring devices, security mechanisms and network controls. This reference implementation will integrate OpenC2 into existing solutions such as OpenStack, OpenFlow, SCAP and (possibly) OpenAM, however ultimately we envision the commercial vendors adapting their applications to become compliant with the OpenC2 standards.

Design considerations

Though our approach to providing machine executable instructions that reflect organizational intent is new, the problem is not therefore we have adopted several design principles that should enable us to avoid some of the pitfalls that plagued earlier activities.

Rather than spend a great deal of time and energy designing a solution that handles every possible situation and edge-case, we recommend first developing an 80% solution to prove the effectiveness of the approach and provide a solid functional and operational baseline. From there, development of solutions for specific niches and edge-cases can proceed.

We will analyze the problem space and determine the small set of verbs that covers the majority of the space as it is currently understood. We will articulate that small set of verbs in a context-specific extensible way. As new attacks or solutions are developed or current niche concerns become more central, we can expand the core language to address these changes, but we must be extremely disciplined and limit extensions to the core as much as possible. In this way we can address the things that matter more efficiently and add complexity only when it is needed to address actual concerns.

To avoid paralyzing handwringing over the issue of whether the device will respond perfectly to commands in all situations, we will design this capability with the following two assumptions in mind:

1. We assume all participating systems are compromised and consider device error as one form of compromise. While it is important that OpenC2 commands lead to correct actions the vast majority of the time, we do not expect or require perfect performance from devices in highly ambiguous situations.
2. We rely on the COTS vendors to make reasonable and effective interpretations of the instruction sets as they relate to the functions their software supports. The orchestrator tells the device what to do, not how to do it. We assume the device knows how to do what it is supposed to do and do not assign a babysitting role to the orchestrator.

This allows us to get highly assured behavior over the bulk of the problem space while avoiding the complexity needed to gain assurance at the fringes.

We can operate effectively with a less-than-perfect language and less-than-perfect device behavior because OpenC2 addresses coherent system behavior and represents a small step in the direction of self-healing systems. The point of the orchestrator is to address deficiencies in the system by coordinating the actions of other parts of the system. The system instrumentation that provoked the action in the first place provides indications as to whether the response addressed the problem and provokes corrective measures in cases when it did not.

Addressing Systemic Issues

Facing the proprietary business model

Large companies gain market advantage by offering solutions that use a proprietary orchestration language and that support their single-vendor suite of components. Though enterprise consumers see clear value in the ability to buy best of breed solutions, the current market does not facilitate this. The vendor adoption of OpenC2, and therefore its overall usefulness, will be stymied if the open approach cannot show value to vendors that counteracts this strong market force. Vendors who do not currently offer integrated suites clearly gain value from OpenC2. Providing an OpenC2 interface enables them to be more easily integrated into enterprise systems and to participate in courses of action that require capabilities not offered in their individual products. The value to vendors that do offer integrated suites is less easily seen but nonetheless significant. OpenC2 will enable these vendors to offer support for more complex courses of action by incorporating special functionality from another vendor to augment their suite of capabilities. It also enables the interoperation of integrated suites that address different segments of the market. Novel combinations of network management and security management or of host and network monitoring suites or of detection and response technologies become possible. As consumers become more sophisticated in their requirements, such combinations will become more sought after, and OpenC2 can enable integrated solution providers to collaborate to address these needs and avoid a clash of the titans that could compromise their current market share. Carefully constructed reference implementations or use case discussions can highlight this advantage and spur uptake of OpenC2 among vendors with integrated solution suites.

Connecting real devices

The success of a control fabric corresponds directly to the number and variety of devices that can be controlled by it. It is hoped that once the standard is developed, it will become established as a best of breed method and vendors would implement interfaces to their devices that accommodate that standard. Initially, it is unlikely vendors will modify existing devices to make them "OpenC2 compliant" yet we would want to be able to use this existing technology in our orchestrated responses. Some may comply with other standards, and we should be able to integrate with these standards. Others may have non-standard controls, and adaptors or wrappers for those devices would have to be written. Enterprise users would have strong reason to advocate for compliance with, or adaptation to, this open standards approach because it will increase vendor competition and enable them to readily integrate point solutions into a best of breed architecture and avoid costly single vendor suites.

Capturing intent

The success of the control fabric is also predicated upon the ability to translate intent into actionable instructions. For decisions to be useful in this context they need to express a finite action or set of actions that is to be performed given a finite set of recognizable conditions. This requires a good amount of verbal "engineering" to get from the original statement of desired objective to unambiguous if-then statements that can be translated into machine readable instructions. Intent can be large; the expression of that intent needs to be decomposed into actionable parts. The intent to "protect against data leaks" can be

decomposed into a set of decisions: protect data at rest, protect data in motion, protect data in portable devices, and so on. Decisions are often expressed as "no's;" these "no's" need to be translated into actionable responses that draw out the unexpressed consequences of violating the decision. Instead of stating "X is not allowed," the decision maker must be able to say "if X happens, do Y to protect me." While our activities will not provide any technology to address these challenges, our capability will be engineered with an understanding of this complexity and will be resilient to input errors. We will also provide documentation to help decision-makers understand the logic of executable decisions and templates to facilitate executable statements of intent.

Conclusion

We propose to develop an open source system command and control capability to address the central problem of real-time active defense, that of orchestrating device actions. The existence of such a capability should spur innovations above and below this orchestration level and enable progress toward resilient systems that respond to threat in real-time.

Addendum: Understanding the Place of the Core Language in Translating Human Decisions to Machine Action

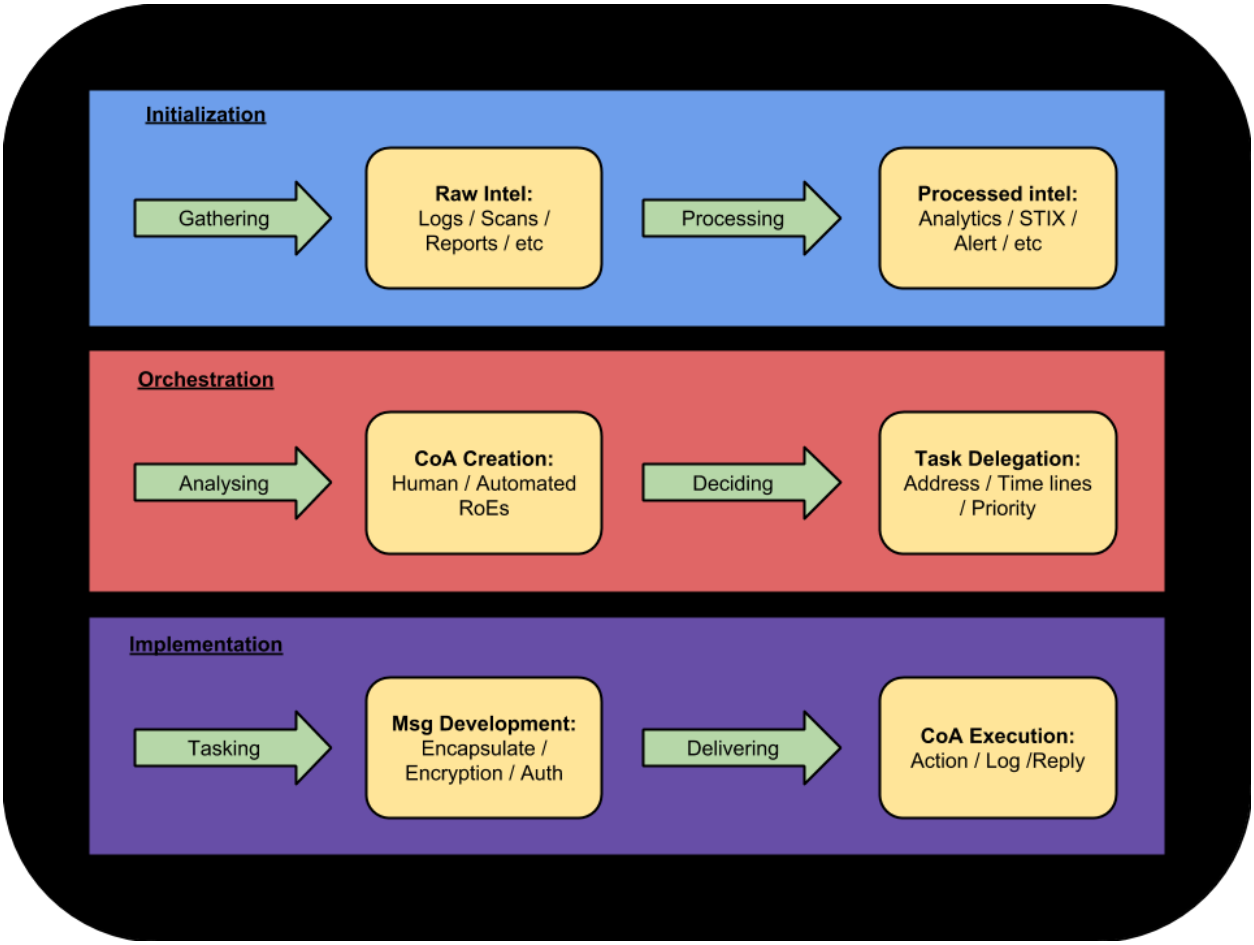

Figure 1: A notional hierarchical relationship between Intent and Execution

Figure 1 depicts a high level approach to connect analysis with courses of action. As organizations generate or receive new threat indicators, they must decide how to best respond to them. This decision process includes the consideration of any predetermined Rules of Engagement (RoEs), and results in the tasking of commercial solutions with the intent to enable the dynamic orchestration of computer network defenses as they relate to sensors, host configurations, and data services in an effort to create resilient networks and reduce adversarial effectiveness. These tasks are expressed in an action-oriented, machine-readable language and delivered via messages using a messaging protocol that provides encryption and authentication.

The process as described above is a human-in-the-loop process. Humans decide what course of action is appropriate in a given situation and what observations indicate that a particular situation is in play. OpenC2 supports the automation of the execution of those human decisions. The human picks the course of action and the triggers; OpenC2 recognizes the triggers and

distributes the component activities to the various parts of the system. However, OpenC2 could also accelerate the move to autonomic systems by facilitating the execution of machine course of action decisions based on observations of system state. This is a topic for future investigation.