Akraino Security Requirements 0.2-draft

Introduction

The Akraino Security Requirements document is a list security items created by the Akraino security sub-committee. This document includes security best practices/requirements identified by the ONAP project (also a Linux Foundation project) which are also common to the Akraino project.

Runtime Security

General Security

This section provides details on the Akraino general security requirements on various security areas such as user access control, network security, ACLs, infrastructure security, and vulnerability management. These requirements cover topics associated with compliance, security patching, logging/accounting, authentication, encryption, role-based access control, least privilege access/authorization.
Integration and operation within a robust security environment is necessary and expected. The security architecture will include one or more of the following: IDAM (Identity and Access Management) for all system and application access, network vulnerability scans, OS, Database and application patching, malware detection and cleaning, DDOS prevention, network security gateways (internal and external) operating at various layers, host and application based tools for security compliance validation, aggressive security patch application, tightly controlled software distribution and change control processes and other state of the art security solutions. Akraino is expected to function reliably within such an environment and the developer is expected to understand and accommodate such controls and can expected to supply responsive interoperability support and testing throughout the product's lifecycle.
Akraino **MUST** implement and enforce the principle of least privilege on all protected interfaces.
Akraino **MUST** provide a mechanism (e.g., access control list) to permit and/or restrict access to services on Akraino by source, destination, protocol, and /or port.
Akraino **SHOULD** provide a mechanism that enables the operators to perform automated system configuration auditing at configurable time intervals.Note: Probably some security architecture work needed
Akraino **SHOULD** provide the capability for the Operator to run security vulnerability scans of the operating system and all application layers.
Akraino **MUST** support network segregation on Akraino external network interfaces: separation of O&M traffic from other traffic. Also, further separation like DB traffic, traffic between VNFs and Akraino, … TBD. The separation is realized in the infra using technologies like VLAN, VXLAN, VPN.- note: probably some security architecture work needed Akraino **SHOULD** support network segregation on Akraino internal interfaces, between and inside the Kubernetes clusters: separation of O&M traffic from other traffic. The separation is realized eg. using network namespaces and K8s network policies.Notes: - probably some security architecture work needed- K8s network policies: the level of support depends on the chosen CNI plugin
Note: Modified the original VNF req – still a draft: Akraino **SHOULD** support the use of HW rooted security technologies like HSM, SGX, virtual TPM for protection of more critical data (like encryption keys, secrets).Notes:- Security architecture work needed: for which use cases in Akraino are these technologies planned/possible to be applied. One example: - The limitation with usage of TPM, vTPM and SGX: not feasible to use for a workload that can migrated between CPUs/machines (= the typical way to deploy in cloud).   - HSM does not have this limitation as it is accessed over network protocol
Akraino Provider **MUST** have patches available for vulnerabilities in Akraino as soon as possible. Patching shall be controlled via change control process with vulnerabilities disclosed along with mitigation recommendations.
Akraino **MUST** support encrypted access protocols, e.g., TLS, SSH, SFTP.
Akraino **MUST** store Authentication Credentials used to authenticate to other systems encrypted except where there is a technical need to store the password unencrypted in which case it must be protected using other security techniques that include the use of file and directory permissions. Ideally, credentials SHOULD rely on a HW Root of Trust, such as a TPM or HSM.
For all GUI and command-line interfaces, Akraino **MUST** provide the ability to present a warning notice that is set by the Operator. A warning notice is a formal statement of resource intent presented to everyone who accesses the system.
Akraino **MUST** allow the Operator to disable or remove any security testing tools or programs included in Akraino, e.g., password cracker, port scanner.
Akraino **MUST** support the ability to prohibit remote access to Akraino via a host based security mechanism.
Akraino **MUST** log any security event required by Akraino Requirements to Syslog using LOG_AUTHPRIV for any event that would contain sensitive information and LOG_AUTH for all other relevant events.
Akraino **MUST** be operable without the use of Network File System (NFS).
Akraino **MUST NOT** contain any backdoors.
If SNMP is utilized, Akraino **MUST** support at least SNMPv3 with message authentication.
Akraino application processes **MUST NOT** run as root.
Login access (e.g., shell access) to the operating system layer, whether interactive or as part of an automated process, **MUST** be through an encrypted protocol such as SSH or TLS.
Akraino **MUST**, after a successful login at command line or a GUI, display the last valid login date and time and the number of unsuccessful attempts since then made with that user's ID. This requirement is only applicable when the user account is defined locally in Akraino.
Akraino **MUST** include a configuration that specifies the targeted parameters, e.g. a limited set of ports, over which Akraino will communicate (including internal, external and management communication). # Need to communicate this to the API subcommittee. #

Identity and Access Management

Akraino **MUST**, if not integrated with the Operator's Identity and Access Management system, support the creation of multiple IDs so that individual accountability can be supported.
Akraino **MUST** allow the Operator to restrict access based on the assigned permissions associated with an ID in order to support Least Privilege (no more privilege than required to perform job functions).
Each architectural layer of Akraino (eg. operating system, network, application) **MUST** support access restriction independently of all other layers so that Segregation of Duties can be implemented.
Akraino **MUST NOT** allow the assumption of the permissions of another account to mask individual accountability. For example, use SUDO when a user requires elevated permissions such as root or admin.
Akraino **MUST** set the default settings for user access to deny authorization, except for a super user type of account. When Akraino is installed, nothing should be able to use it until the super user configures Akraino to allow other users (human and application) have access.
Akraino **MUST** support strong authentication, also known as multifactor authentication, on all protected interfaces exposed by Akraino for use by human users. Strong authentication uses at least two of the three different types of authentication factors in order to prove the claimed identity of a user. # Look at making this MUST for infrastructure and SHOULD for other areas #
Akraino **MUST** disable unnecessary or vulnerable cgi-bin programs. # Completely disallow cgi-bin #
Akraino **MUST** provide access controls that allow the Operator to restrict access to Akraino functions and data to authorized entities. # Least privilege #
Akraino **SHOULD** support OAuth 2.0 authorization using an external Authorization Server.
Akraino **MUST**, if not integrated with the Operator's Identity and Access Management system, support configurable length and password expiration.
Akraino **MUST**, if not integrated with the Operator's Identity and Access Management system, support Role-Based Access Control to enforce least privilege.
Akraino **MUST**, if not integrated with the Operator's Identity and Access Management system, comply with "password complexity" policy. When passwords are used, they shall be complex and shall at least meet the following password construction requirements: (1) be a minimum configurable number of characters in length, (2) include 3 of the 4 following types of characters: upper-case alphabetic, lower-case alphabetic, numeric, and special, (3) not be the same as the UserID with which they are associated or other common strings as specified by the environment, (4) not contain repeating or sequential characters or numbers, (5) not to use special characters that may have command functions, and (6) new passwords must not contain sequences of three or more characters from the previous password.
Akraino MUST not store authentication credentials to itself in clear text or any reversible form and must use salting.
Akraino **MUST**, if not integrated with the Operator's Identity and Access Management system, support the ability to disable the userID after a configurable number of consecutive unsuccessful authentication attempts using the same userID.
Akraino **MUST**, if not integrated with the Operator's identity and access management system, authenticate all access to protected GUIs, CLIs, and APIs. Note: I (could) read this like: "if I integrate with operator's IdAM then I don't need to do anything on authentication". Maybe improve wording…
Akraino **MUST** integrate with standard identity and access management protocols such as LDAP, TACACS+, Windows Integrated Authentication (Kerberos), SAML federation, or OAuth 2.0.
Akraino **MUST** have the capability of allowing the Operator to create, manage, and automatically provision user accounts using an Operator approved identity lifecycle management tool using a standard protocol.
Akraino **MUST** support account names that contain at least A-Z, a-z, 0-9 character sets and be at least 6 characters in length.
A failed authentication attempt **MUST NOT** identify the reason for the failure to the user, only that the authentication failed.
Akraino **MUST NOT** display "Welcome" notices or messages that could be misinterpreted as extending an invitation to unauthorized users.
Akraino **MUST** provide a means for the user to explicitly logout, thus ending that session for that authenticated user.
Akraino **MUST**, if not integrated with the Operator's Identity and Access Management system, or enforce a configurable "terminate idle sessions" policy by terminating the session after a configurable period of inactivity.

API Security

This section covers API security requirements when these are used by the Akraino. Key security areas covered in API security are Access Control, Authentication, Passwords, PKI Authentication Alarming, Anomaly Detection, Lawful Intercept, Monitoring and Logging, Input Validation, Cryptography, Business continuity, Biometric Authentication, Identification, Confidentiality and Integrity, and Denial of Service.
The solution in a virtual environment needs to meet the following API security requirements:
Akraino **SHOULD** integrate with the Operator's authentication and authorization services (e.g., IDAM).Note: Should specify explicitly what has to be supported.
Akraino **MUST** implement the following input validation control: Check the size (length) of all input. Do not permit an amount of input so great that it would cause Akraino to fail. Where the input may be a file, Akraino API must enforce a size limit.
Akraino **MUST** implement the following input validation controls: Do not permit input that contains content or characters inappropriate to the input expected by the design. Inappropriate input, such as SQL expressions, may cause the system to execute undesirable and unauthorized transactions against the database or allow other inappropriate access to the internal network (injection attacks).
Akraino **MUST** implement the following input validation control on APIs: Validate that any input file has a correct and valid Multipurpose Internet Mail Extensions (MIME) type. Input files should be tested for spoofed MIME types.

Security Analytics

This section covers Akraino security analytics requirements that are mostly applicable to security monitoring. The Akraino Security Analytics cover the collection and analysis of data following key areas of security monitoring:

- Anti-virus software
- Logging
- Data capture
- Tasking
- DPI
- API based monitoring
- Detection and notification
- Resource exhaustion detection
- Proactive and scalable monitoring
- Closed loop monitoring
- Interfaces to management and orchestration
- Malformed packet detections
- Service chaining

- Dynamic security control
- Dynamic load balancing
- Connection attempts to inactive ports (malicious port scanning)

The following requirements of security monitoring need to be met by the solution in a virtual environment.
Akraino **MUST** support Real-time detection and notification of security events.
Akraino **MUST** support Integration functionality via API/Syslog/SNMP to other functional modules in the network (e.g., PCRF, PCEF) that enable dynamic security control by blocking the malicious traffic or malicious end users.Note: PCRF, PCEF are not good examples here à to be changed or removed
Akraino **MUST** support API-based monitoring to take care of the scenarios where the control interfaces are not exposed, or are optimized and proprietary in nature.
Akraino **MUST** support detection of malformed packets due to software misconfiguration or software vulnerability, and generate an error to the syslog console facility.
Akraino **MUST** support proactive monitoring to detect and report the attacks on resources so that Akraino's and associated VMs can be isolated, such as detection techniques for resource exhaustion, namely OS resource attacks, CPU attacks, consumption of kernel memory, local storage attacks.
Akraino **SHOULD** operate with anti-virus software which produces alarms every time a virus is detected.
Akraino **MUST** protect all security audit logs (including API, OS and application-generated logs), security audit software, data, and associated documentation from modification, or unauthorized viewing, by standard OS access control mechanisms, by sending to a remote system, or by encryption.
Akraino **MUST** log successful and unsuccessful authentication attempts, e.g., authentication associated with a transaction, authentication to create a session, authentication to assume elevated privilege.
Akraino **MUST** log logoffs.
Akraino **MUST** log starting and stopping of security logging.
Akraino **MUST** log success and unsuccessful creation, removal, or change to the inherent privilege level of users.
Akraino **MUST** log connections to the network listeners of the resource.
Akraino **MUST** log the field "event type" in the security audit logs.
Akraino **MUST** log the field "date/time" in the security audit logs.
Akraino **MUST** log the field "protocol" in the security audit logs.
Akraino **MUST** log the field "service or program used for access" in the security audit logs.
Akraino **MUST** log the field "success/failure" in the security audit logs.
Akraino **MUST** log the field "Login ID" in the security audit logs.
Akraino **MUST NOT** include an authentication credential, e.g., password, in the security audit logs, even if encrypted.
Akraino **MUST** detect when its security audit log storage medium is approaching capacity (configurable) and issue an alarm.
Akraino **MUST** support the capability of online storage of security audit logs.
Akraino **MUST** activate security alarms automatically when a configurable number of consecutive unsuccessful login attempts is reached.
Akraino **MUST** activate security alarms automatically when it detects the successful modification of a critical system or application file.
Akraino **MUST** activate security alarms automatically when it detects an unsuccessful attempt to gain permissions or assume the identity of another user.
Akraino **MUST** include the field "date" in the Security alarms (where applicable and technically feasible).
Akraino **MUST** include the field "time" in the Security alarms (where applicable and technically feasible).
Akraino **MUST** include the field "service or program used for access" in the Security alarms (where applicable and technically feasible).
Akraino **MUST** include the field "success/failure" in the Security alarms (where applicable and technically feasible).
Akraino **MUST** include the field "Login ID" in the Security alarms (where applicable and technically feasible).
Akraino **MUST** restrict changing the criticality level of a system security alarm to users with administrative privileges.
Akraino **MUST** monitor API invocation patterns to detect anomalous access patterns that may represent fraudulent access or other types of attacks, or integrate with tools that implement anomaly and abuse detection.
Akraino **MUST** generate security audit logs that can be sent to Security Analytics Tools for analysis.
Akraino **MUST** log successful and unsuccessful access to Akraino resources, including data.
Akraino **MUST** support the storage of security audit logs for a configurable period of time.
Akraino **MUST** have security logging for Akraino applications/services and their OSs be active from initialization. Audit logging includes automatic routines to maintain activity records and cleanup programs to ensure the integrity of the audit/logging systems.
Akraino **MUST** be implemented so that it is not vulnerable to OWASP Top 10 web application security risks.
Akraino **MUST** protect against all denial of service attacks, both volumetric and non-volumetric, or integrate with external denial of service protection tools.
Akraino **MUST** be capable of automatically synchronizing the system clock daily with the Operator's trusted time source, to assure accurate time reporting in log files. It is recommended that Coordinated Universal Time (UTC) be used where possible, so as to eliminate ambiguity owing to daylight savings time.
Akraino **MUST** log the Source IP address in the security audit logs.
Akraino **MUST** have the capability to securely transmit the security logs and security events to a remote system before they are purged from the system.
Akraino **SHOULD** provide the capability of maintaining the integrity of its static files using a cryptographic method.
Akraino **MUST** log automated remote activities performed with elevated privileges.

Data Protection

This section covers Akraino data protection requirements that are mostly applicable to security monitoring.
Akraino **MUST** provide the capability to restrict read and write access to data handled by Akraino.
Akraino **MUST** Provide the capability to encrypt data in transit on a physical or virtual network.
Akraino **MUST** provide the capability to encrypt data on non-volatile memory. Non-volative memory is storage that is capable of retaining data without electrical power, e.g. Complementary metal-oxide-semiconductor (CMOS) or hard drives.
Akraino **SHOULD** disable the paging of the data requiring encryption, if possible, where the encryption of non-transient data is required on a device for which the operating system performs paging to virtual memory. If not possible to disable the paging of the data requiring encryption, the virtual memory should be encrypted.
Akraino **MUST** use NIST and industry standard cryptographic algorithms and standard modes of operations when implementing cryptography.
Akraino **MUST NOT** use compromised encryption algorithms. For example, SHA, DSS, MD5, SHA-1 and Skipjack algorithms. Acceptable algorithms can be found in the NIST FIPS publications (https://csrc.nist.gov/publications/fips) and in the NIST Special Publications (https://csrc.nist.gov/publications/sp).
Akraino **MUST** use, whenever possible, standard implementations of security applications, protocols, and formats, e.g., S/MIME, TLS, SSH, IPSec, X.509 digital certificates for cryptographic implementations. These implementations must be purchased from reputable vendors or obtained from reputable open source communities and must not be developed in-house.
Akraino **MUST** provide the ability to migrate to newer versions of cryptographic algorithms and protocols with minimal impact.
Akraino **MUST** support digital certificates that comply with X.509 standards.Note: Security architecture should define all the use cases for certificates
Akraino **MUST NOT** use keys generated or derived from predictable functions or values, e.g., values considered predictable include user identity information, time of day, stored/transmitted data.
Akraino **MUST** provide the capability of using X.509 certificates issued by an external Certificate Authority.
Akraino **MUST** be capable of protecting the confidentiality and integrity of data at rest and in transit from unauthorized access and modification.Note: Either as part of req, or separately: specify *how* to protect the data; can be different approach for:- keys/secrets- DBs- configuration data- logs- …

Cryptography

This section covers Akraino cryptography requirements that are mostly applicable to encryption or protocol methods.
Akraino **SHOULD** support an automated certificate management protocol such as CMPv2, Simple Certificate Enrollment Protocol (SCEP) or Automated Certificate Management Environment (ACME).Notes:- Possibly also: Akraino should support installing certificates as part of configuration data, ie "offline enrollment"?- For Akraino we could list explicitly which protocols MUST be supported- Security architecture to define that only AAF certman need to support interface to CA (I think)
Akraino **SHOULD** provide the capability to integrate with an external encryption service.Note: Security architecture work: Which use cases, which protocol (s)
Akraino **MUST** use symmetric keys of at least 112 bits in length.
Akraino **MUST** use asymmetric keys of at least 2048 bits in length.
Akraino **MUST** provide the capability to configure encryption algorithms or devices so that they comply with the laws of the jurisdiction in which there are plans to use data encryption.
Akraino **MUST** provide the capability of allowing certificate renewal and revocation.
Akraino **MUST** provide the capability of testing the validity of a digital certificate by validating the CA signature on the certificate.
Akraino **MUST** provide the capability of testing the validity of a digital certificate by validating the date the certificate is being used is within the validity period for the certificate.
Akraino **MUST** provide the capability of testing the validity of a digital certificate by checking the Certificate Revocation List (CRL) for the certificates of that type to ensure that the certificate has not been revoked.
Akraino **MUST** provide the capability of testing the validity of a digital certificate by recognizing the identity represented by the certificate - the "distinguished name".
Akraino **MUST** support HTTP/S using TLS v1.2 or higher with strong cryptographic ciphers.
Akraino **MUST** support the use of X.509 certificates issued from any Certificate Authority (CA) that is compliant with RFC5280, e.g., a public CA such as DigiCert or Let's Encrypt, or an RFC5280 compliant Operator CA.Note: Akraino provider cannot require the use of self-signed certificates in an Operator's run time environment.

Code Security

CII Badging Program

What is the CII Badging Program?

- CII (Core Infrastructure Initiative) Badge may be achieved by the projects which follow the Best practices criteria for Free/Libre and Open Source Software (FLOSS).
- CII has been created by the Linux Foundation in response to previous security issues in open-source projects (e.g. Heartbleed in OpenSSL).
- The CII Badging is associated to the areas as follows:

Basics, Change Control, Reporting, Quality, Security & Analysis

- Projects in Akraino should be CII certified to an appropriate level to confirm with expectation of carrier grade.

CII Badging Levels
There are 3 CII Badging levels which are as follows:

• Passing

• Silver
• Gold

When a new project starts the badging process, they will begin at 0% completeness and as they progress the % will increase.
To see a list of all Akraino projects and their level of completions refer to link: <link to be provided>
Akraino CII Compliance Levels
For Akraino, 4 levels (Akraino Compliance) of compliance have been defined:
For each Akraino compliance level, all the projects in Akraino should comply to certain standards when it comes to CII badging.
Akraino Level 1: 70 % of the code in Gerrit must have an 100% completion in the CII badging towards passing levelwith the non-passing projects reaching 80% completion in the CII badging towards passing level Non-passing projects MUST pass specific cryptography criteria outlined by the Security Subcommittee*
Akraino Level 2: 70 % of the projects in Gerrit must have an 100% completion in the CII badging for silver level with non-silver projects completed passing level and 80% completion towards silver level
Akraino Level 3: 70% of the projects in Gerrit must have an 100% completion in the CII badging for gold level with non-gold projects achieving silver level and achieving 80% completion towards gold level
Akraino Level 4: 100 % passing gold.
Some of the important high level *example* criteria associated to the various levels are listed as follows for quick reference:

Level	Example Details/Criteria
Passing	The project website MUST succinctly describe what the software does (what problem does it solve?).The project MUST use at least one automated test suite that is publicly released as FLOSS (this test suite may bemaintained as a separate FLOSS project).
Silver	The project MUST document what the user can and cannot expect in terms of security from the software producedby the project. The project MUST identify the security requirements that the software is intended to meet and anassurance case that justifies why these requirements are met. The assurance case MUST include: a description of the threat model, clear identification of trust boundaries, and evidence that common security weaknesses have beencountered
Gold	The project MUST have at least 50% of all proposed modifications reviewed before release by a person other thanthe author, to determine if it is a worthwhile modification and free of known issues which would argue against itsinclusion.

Badge Specific Adherence requirements
Each of the Badging level is associated with compliance requirements which in turn may vary from being e.g. absolute to being as varied as recommendatory in nature.
The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in the Badging guideline documents are to be interpreted as below. (The terms are similar to what is described in RFC2119/RFC8174).

- The term MUST is an absolute requirement, and MUST NOT is an absolute prohibition.
- The term SHOULD indicates a criterion that is normally required, but there may exist valid reasons in particular circumstances to ignore it. However, the full implications must be understood and carefully weighed before choosing a different course.
- The term SUGGESTED is used instead of SHOULD when the criterion must be considered, but valid reasons to not do so are even more common than for SHOULD.
- Often a criterion is stated as something that SHOULD be done, or is SUGGESTED, because it may be difficult to implement or the costs to do so may be high.
- The term MAY provides one way something can be done, e.g., to make it clear that the described implementation is acceptable.
- To obtain a badge, all MUST and MUST NOT criteria must be met, all SHOULD criteria must be met OR the rationale for not implementing the criterion must be documented, and all SUGGESTED criteria have to be considered (rated as met or unmet). In some cases a URL may be required as part of the criterion's justification.

Code Scans

Akraino **SHOULD** have source code scanned using scanning tools (e.g., Fortify) and provide reports.
Note: The following is one of the most important-ones: Akraino **MUST** have all code (e.g., QCOW2) and configuration files (e.g., HEAT template, Ansible playbook, script) hardened, or with documented recommended configurations for hardening and interfaces that allow the Operator to harden Akraino. Actions taken to harden a system include disabling all unnecessary services, and changing default values such as default credentials and community strings.
Note: Traffic type separation reqs, I modified the original VNF req & split into 2: **Req 1)**
Recommedations

- Use Coverity Scan https://scan.coverity.com/ to perform static code scans on all Akraino code.
- Automate scanning by enabling Jenkins to trigger weekly scans with Coverity Scan.

- Deliver scan reports to the PTLs (Project Technical Lead) for each project PTLs will be responsible for getting the vulnerabilities resolved (fixed or designated as false positive).
- All projects in a release must have the high vulnerabilities resolved by MS-3.
- All projects in a release must have the high and medium vulnerabilities resolved by MS-4.
- The Security Committee will host session to help projects walk through the scanning process and reports.

Next Steps

- Review the OPNFV scanning process at https://wiki.opnfv.org/display/security/Security+Scanning to see if it can be adopted as the Akraino static code scanning process.

Tools that have been assessed: Coverity Scan (LF using the tool in OPNFV and other projects), HP Fortify (AT&T evaluation), Checkmarx (AT&T evaluation), Bandit (AT&T evaluation)
Preliminary Decision: Coverity Scan https://scan.coverity.com/
Description: Coverity Scan is a service by which Synopsys provides the results of analysis on open source coding projects to open source code developers that have registered their products with Coverity Scan. Coverity Scan is powered by Coverity® Quality Advisor. Coverity Quality Advisor surfaces defects identified by the Coverity Static Analysis Verification Engine (Coverity SAVE®). Synopsys offers the results of the analysis completed by Coverity Quality Advisor on registered projects at no charge to registered open source developers.
Open Source use: 4000+ open source projects use Coverity Scan
Languages supported: C/C++, C#, Java, Javascript, Python, Ruby
Current Activity:
Coverity Scanning Process:
Coverity static analysis works by instrumenting through build capture. The components which make up the Akraino project can be managed a number of ways:

- If the Akraino project can be built from source in a single command, then Coverity can to create component maps.
- If the separate components are built individually, then each component can be submitted as a separate project.
- Coverity recommends storing the projects in a hierarchical structure in Github with the Akraino parent project referring to the project (i.e. Akraino /component_name). There are a few projects already in Scan which follow this structure. (is Akraino stored this way?) Each Akraino project has its own hierarchy in Gerrit (its own Git tree). Can they do a Git Pull, Git Clone on an arbitrary git repository?

Restrictions on builds: (from https://scan.coverity.com/)

Maximum Lines of Code in Project	**Frequency of Scans**
<100K lines of code	Up to 28 builds per week, with a maximum of 4 builds per day
100K to 500K lines of code	Up to 21 builds per week, with a maximum of 3 builds per day
500K to 1 million lines of code	Up to 14 builds per week, with a maximum of 2 build per day
>1 million lines of code	Up to 7 builds per week, with a maximum of 1 build per day

Once a project reaches the maximum builds per week, additional build requests will be rejected. The submitter will be able to re-submit the build request the following week.
Scan is self-service: Coverity provides the analysis infrastructure and results, but the submitter must provide the instrumented artifacts to analysis. Scan provides integration with TravisCI/Github.
To use Scan, the submitters will have to create an account and submit their project at https://scan.coverity.com/projects
Coverity requires a code contributor to submit a project because of their responsible disclosure process for issues the tool may identify within the code.
Next Steps:

Image Signing/Verification