GETS Engineering and Development Center

Terpene Overview

1.0-BETA

(U) Table of Contents

(U) Table of Figures....3
(U) Points of Contact....3
(U) Changes....3
(U) Purpose....4
(U) Intended Audience....4
(U) References....4
(U) Background....4
(U) Conceptual Overview....5
(U) Sequential Event Streams....5
(U) Definitive Data Record....5
(U) 5
(U) Terpene Framework....6
(U) Terpene Components....6
(U) Service Components....6
(U) Application Components....6
(U) Terpene Component Execution Environment (PaaS)....7
(U) Terpene Service Management 7
(U) Conclusion....8
(U) Case Studies....8
(U) Cloud-Scalability....8
(U) AO/DO....8
(U) Integral Traceable Audit Log....8
(U) Repair Data 8
(U) Projection Modeling and War Gaming....8
(U) Future Proof: New Analytical Processing Models....8

(U) Table of Figures

(U) Points of Contact

	UNCLASSIFIED	SIPRNet	JWICS
Technical	terpene-devs@geoint.org	developers@agi.nga.smil.mil	developers@geoint.ic.gov
Government/Policy	gets-mgmt@mail.mil	gets-mgmt @agi.nga.smil.mil	gets-mgmt@geoint.ic.gov

(U) Table 0: Points of Contact Table

(U) Changes

DATE	CHANGED BY	DETAILS	APPROVED BY
7Jul2015	steve@t-3-solutions.com	Initial draft	

(U) Table 0: Document Changes

(U) Purpose

(U) Terpene is a secure, distributed, service-oriented framework and component execution environment designed specifically for the unique needs of the US military, abstracting away cross-cutting concerns ranging from Information Assurance/STIG requirements to Autonomous Operations/Disconnected Operations (AO/DO), allowing developers to focus purely on the functional aspects of their software. This document provides a high-level introduction to both the terpene framework and Platform as a Service (PaaS) execution environment, highlighting their impact to time-to-market, ability for the software to adapt to changing mission requirements, inherent security and regulatory compliance features, and enablement of true "Space to Mud" information systems. Finally, mission-oriented and disaster recovery case studies will be briefly discussed to demonstrate the versatility of the framework.

(U) Most of the significant terpene features are possibly by leveraging a sequential event persistence system, Canon. This document will also provide the necessary context to understand the concepts, and complementary nature, of the two systems.

(U) Intended Audience

(U) The intended audiences for this document are mission managers, project managers, security and information assurance professionals, and developers new to the terpene platform.

(U) References

- *Introducing Canon Event Database (GEOINT-CANON-2015-01)*
- *Terpene "Compliance as a Service" (GEOINT-TERPENE-2015-09)*

(U) Background

(U) Terpene was developed by the US Army/G2 in response to complex, often competing, requirements for government-developed software required to operate on high-security and diverse environments. All such software must undergo multiple security accreditation processes that operate with different Information Assurance (IA) policies, each requiring different types and formats of documentation as well as complex change management processes. This administrative burden of creating software in this environment alone is extremely high, and that is before the non-functional and functional software requirements stemming from the unique environmental requirements (not to mention the actual business requirements), and how these may conflict with these policies. Terpene was conceived to help remedy these issues by satisfying common requirements (IA, environmental, etc) centrally, not in each program, resulting in less paperwork, faster time to market, less risk, and increased contractual competition.

(U) The Army/G2 realized that these burdens are a significant overhead for each program beyond the Army, but one that is shared between every DoD, IC, and possibly wider agencies. This overhead translates to not only significant program risk and costs on behalf of the government, but also to any potential contract associated with such a project. These risks and costs stifles competition by smaller, innovative, companies – talent that the government needs to attract. As such, the Army/G2 developed terpene as an Open Source software to not only promote the adoption of the project across the government, but also to invite community involvement in the development and maintenance of the project, increasing the pool of potential contract bidders and reducing costs.

(U) Conceptual Overview

(U) Sequential Event Streams

(U) Definitive Data Record

(U)

(U) Terpene provides a secure service-based execution environment promoting the development of smaller component-based software by abstracting common cross-cutting concerns[1] to services provided by the terpene platform (see Figure 1). Abstracting cross-cutting concerns away from components written to run on the terpene platform typically results in less lines of code by not need to include repetitive code blocks or handle low-level resource concerns. By centralizing each of these concerns in their own service component we further reduce repetitive code. The result of all this compartmentalization is code that is much easier to produce and maintain, resulting in less flaws (bugs). Removing repetitive code results in much more than just productivity improvements as services can ensure that cross-cutting concerns such as authentication, authorization, auditing, data validation/filtering, high-availability, fault-tolerance, etc are managed centrally by the service component instead of requiring software components to handle these concerns individually. By not having to reproduce those aspects in each software component, terpene ensures that those concerns are always addressed at runtime.

(U) In addition to the productivity and inherent security benefits of the terpene service-based architecture, core terpene services are designed to provide a consistent runtime environment for application components regardless of the deployment architecture. This consistency allows the software components, without code changes, to execute on diverse environments. For example, components written to execute on a terpene platform can execute on platforms ranging from an embedded device such as a Raspberry Pi to a Platform as a Service (PaaS) or Infrastructure as a Service (IaaS) cloud, such as Amazon AWS, without any code modifications. Similarly, these

1 Cross-cutting concerns are software requirements that are not core to the problem domain (business concern) but are necessary for the software to function. These concerns are often common across multiple software applications, such as logging, auditing, monitoring, accessing external resources, etc.

same applications can run on a fault-tolerant, highly-available network or on a disconnected or disadvantaged network, again, without modifications to the application component. This flexibility is achieved through terpene components simply accessing these services though their publicly defined software interfaces (Figure 2), and specific service component implementations abstracting away the platform or environmental concerns (see Figure 3).

(U) Terpene Framework

(U) Terpene Components

(U) Service Components

(U) Application Components

(U) Components are conceptual units of deployment within terpene, normally comprised of one or more classes. Components are added to terpene by through the deployment of standard java archives (jar) or zip files. A single jar or zip can contain multiple terpene components, including business logic components, service definitions, and service implementations. Typically, however, developers will want to deploy each component individually (for more information, see the Terpene Developer Guide). What differentiates a terpene component from standard java class is the ability for terpene to discover (recognize) and then manage the lifecycle of a class as a terpene component, which is defined by the terpene services themselves. Since new service components can be created and deployed with the same ease as a standard terpene component, this becomes a very powerful concept (see Figure 5).

(U) Terpene platforms guarantee that components deployed on terpene cannot access code, either at rest or on the heap (bytecode), used by another component. To support this, code deployed to terpene must contain all of its dependencies (a common practice called an “uber jar”; further discussed in the Terpene Developer Guide). Components must include all of their dependencies because all terpene components are deployed in isolation, and cannot access shared libraries beyond the JVM (which is uniquely protected by the JVM). This guaranteed isolation addresses security concerns partially addressed in several STIG findings (V-26949, V-6149, V-35224), though terpene takes this one step further by ensuring that components cannot change the runtime bytecode shared between components. For information on how a specific terpene platform implementation achieves secure execution isolation you must consult the platform-specific documentation.

(U) All terpene components, by default, must be digitally signed, which are verified by the individual terpene platform before a component is loaded, and continues to be protected by ensuring the code is not manipulated once deployed. This ensures that all components deployed on terpene comply with the STIG finding V-6161. Software digital signatures are notoriously complex and easy to get wrong, and complex deployment scenarios (multiple isolated networks, multiple software instances, offline deployments) make this even more difficult. To ease the burden, not only does terpene handle the verification centrally, removing the responsibility from

software components, but terpene also provides convenience tools to support "just in time" digital signatures by authorized administrators. For more information, and recommended best practices for using software digital signatures, please see the Terpene Deployment Guide.

(U) Terpene Component Execution Environment (PaaS)

(U) Terpene Service Management

(U) Terpene service management ensures that components are not deployed to platforms that cannot support its service dependencies. Terpene uses this same service management meta to provide service watchdog features, providing advanced failover/failback service capabilities. During component deployment dependencies on terpene services is verified, and a deployment will fail if the service is not available at the time of deployment. If, however, the service becomes available after a component is deployed, (ie terpene detects that a service has entered a failed state), terpene can attempt various failover/failback strategies to resolve the failure prior to interrupting component execution. These watchdog services are transparent to consuming components, abstracting concerns such as service fault-tolerance away from component responsibilities.

(U) Terpene watchdog services not only attempt to rectify a problem with a service, interrupting a component as a last resort, the watchdog may optionally be configured to fail over to an alternative service implementation until such time that the primary service is once again online, when the service will fail-back. This capability can be used in interesting ways, defined at deployment time by platform configuration. For example, a terpene platform can be configured to support online/offline operations, with automatic failover/failback, by defining alternative service implementations that are only used when a service that require network connectivity has entered failed state (see Figure 6). For more information on how to configure the watchdog services, see the Terpene Deployment Guide. For best practices on creating "alternative" services, including an offline service example, see the Terpene Developer Guide.

(U) So far we have discussed service abstraction based on cross-cutting concerns of component requirements and how developers can leverages this to achieve unprecedented productivity, security, and high-availability capabilities. Terpene service abstraction serves an additional, less apparent but nonetheless important role in allowing components to be deployed in vastly different environments without requiring changes to component code. Combining different service implementations allow a terpene platform to make use of available environmental resources (such as cloud services) or manage limited resources through quotas (which might be necessary when running on a low-power/embedded computer). A specific combination of services, and their configuration, is a "terpene platform", where components are deployed. Terpene ships with two platforms consisting of all required and common core services, called the terpene-platform-local (non-networked) and terpene-platform-cluster (networked). More information about these platforms can be found in their platform-specific documentation. New

platforms are easily created by combining and configuring services in different ways to meet the environmental requirements. Perhaps the easiest way to create a new terpene platform is to simply remove an unneeded service from one of the provided platforms and repackaging (zip and rename) (see Figure 7).

(U) Service definitions and service implementation/configuration is what makes a terpene platform special and capable of leveraging (or operating in spite of) the environment. Again, because terpene components do not access service implementations directly, components are unaware of how a service achieves the requested operation. For example, a web application component, accessing required resources through terpene services, can be deployed without code changes to a laptop suffering from poor network connectivity (disadvantaged network) and a robust cloud service (see Figure 8).

(U) Conclusion

(U) Terpene as a software framework, based simply on services and components, is conceptually simple and not necessarily novel. However, though this simplicity, terpene is uniquely capable of abstracting away cross-cutting concerns, such as security (IA) and environmental requirements, from each software application. The benefits of terpene are realized both in positive impacts to the overall security and productivity of an application as well as in the ability of meeting non-functional requirements of the environment, be it technical or regulatory.

(U) Case Studies

(U) Cloud-Scalability

…Asynchronous and Fault Tolerant, eventual consistency with asynchronous reconciliation (which doubles as the ability to sanitize)

(U) AO/DO

(U) Integral Traceable Audit Log

(U) Repair Data

(U) Projection Modeling and War Gaming

(U) Future Proof: New Analytical Processing Models