NGCRC Project Proposal

Blockhub: Blockchain-based Secure Cross-domain Software Development and Sharing System

Aug 27, 2017

Prepared for
The IS Sector Investment Program

Prepared by
Bharat Bhargava
CERIAS, Purdue University

As of 6/03/2011
Table of Contents

1 Executive Summary .............................................................................................................. 3
  1.1 Abstract .......................................................................................................................... 4
  1.2 Graphical Illustration ....................................................................................................... 5
2 Description of Project ........................................................................................................... 7
  2.1 Statement of Problem ...................................................................................................... 7
  2.2 State of Current Technology and Competition .............................................................. 7
  2.3 Proposed Solution and Challenges ................................................................................ 9
    2.3.1 Blockchain-based software sharing ......................................................................... 10
    2.3.2 Decentralized blockchain-based software development ............................................ 12
    2.3.3 Efficient blockchain storage ..................................................................................... 13
    2.3.4 Challenges of blockchain technology deployment .................................................... 13
    2.3.5 Software Spillage Remediation ................................................................................ 14
    2.3.6 Data Analytics for Software Spillage Detection ....................................................... 16
  2.4 Distinctive Attributes, Advantages, and Discriminators ................................................ 17
    2.5.1 Software ................................................................................................................... 18
    2.5.2 Documentation ......................................................................................................... 18
  2.5 Technical Merit and Differentiation .................................................................................. 19
3 Project Milestones ................................................................................................................ 20
  3.1 Statement of Work .......................................................................................................... 20
    3.1.1 Integration with NGCRC and NGC IR&D Projects .................................................... 22
  3.2 Milestones and Accomplishments .................................................................................. 22
4 Project Budget Estimate ...................................................................................................... 23

List of Tables

Table 1: Executive Summary .................................................................................................. 3
Table 2: Project Budget Estimate ........................................................................................ 24
# Executive Summary

<table>
<thead>
<tr>
<th>Title</th>
<th>Blockhub: Blockchain-based secure cross-domain software development and sharing system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Bharat Bhargava</td>
</tr>
<tr>
<td>Project Lead</td>
<td>Bharat Bhargava</td>
</tr>
<tr>
<td>University</td>
<td>Purdue University</td>
</tr>
<tr>
<td>Requested Funding Amount</td>
<td>$149,999</td>
</tr>
<tr>
<td>Period of Performance</td>
<td>September 1, 2017 - August 31, 2018</td>
</tr>
<tr>
<td>Is this an existing Investment Project?</td>
<td>Yes</td>
</tr>
<tr>
<td>TRL Level of Project</td>
<td>3</td>
</tr>
<tr>
<td>Key Words</td>
<td>Blockchain, data provenance, software spillage, data analytics, cross-domain software sharing, privacy</td>
</tr>
<tr>
<td>Key Partners &amp; Vendors</td>
<td>Context-based Adaptable Defense Against Collaborative Attacks in SOA; End-to-End Security Policy Auditing and Enforcement in Service-Oriented Architecture; Monitoring-Based System for E2E Security Auditing and Enforcement in Trusted and Untrusted SOA; Privacy-Preserving Data Dissemination and Adaptable Service Compositions in Trusted &amp; Untrusted Cloud; “WAXEDPRUNE” (Web-based access to encrypted data processing in untrusted environments) prototype; Secure/Resilient Systems and Data Dissemination/Provenance</td>
</tr>
</tbody>
</table>
1.1 Abstract

To ensure integrity, trust, immutability, and authenticity of software and information (cyber data, user data, and attack event data) in a collaborative environment, research is needed for cross-domain software development, global software collaboration, sharing, access auditing and accountability. Blockchain technology can significantly automate the export auditing and tracking processes and be used for data provenance, trust, and analytics. Provenance allows to determine the quality of data and enables accountability. For effective data forensics/provenance, the identity determination of those who have accessed/updated/disseminated the sensitive cyber data or sensitive software is needed. Once software is made available to parties with the proper authorization level, transactions may occur that spill the software to unauthorized parties. Research is needed to track and control what software components are shared between entities across multiple security domains. It needs to be guaranteed that unauthorized transaction cannot be repudiated. Research is needed to design and develop mechanisms that allow fast detection of software spillages and updates of the trust level of parties involved in such misdemeanor. The main shortcomings of existing techniques for data provenance are that they involve heavyweight processing and storage of metadata and do not provide interoperability across domains. There is a need for developing a distributed data storage mechanism using blockchains which provides tracking of data provenance and software spillage prevention with automated evaluation of the trustworthiness of both the software and parties sharing it. Research is required on how blockchain can increase the trust for software components developed on sites with lower classification level and moved to high side or vice versa.

One goal of this effort is to contribute to NGC IR&D projects and closely collaborate with other NGCRC projects to achieve a showcase system and develop principles of science of security. Specifically, proposed research will focus on blockchain-based software management system ‘Blockhub’ with smart contracts that provides high assurance and highly automated secure cross-domain software sharing and development. It will support a software supply chain that allows to develop and disseminate software modules across multiple departments of a company, providing access auditing and tracking capabilities. ‘Blockhub’ will be built on top of the existing ‘WAXEDPRUNE’ (Web-based Access to Encrypted Data Processing in Untrusted Environments) prototype that was demonstrated at NGC TechExpo in June 2016 and at NGCRC Symposium in April, 2017. The research was published in IEEE Cloud 2017 [1] with Donald Steiner, Leon Li, Jason Kobes, Harry Halpin (MIT) as co-authors, received best poster award [4] at CERIAS Security Symposium 2015 and Purdue University Computer Science Corporate Partners award in April 2017.

We will develop science, experiment and demonstrate the capabilities of Blockhub and work closely with NGCRC researchers (Prof. Kate at Purdue, Harry Halpin at MIT) and NG researchers and engineers to fulfill the needs of potential customers (agencies such as IRS, SSA, AFRL, DoD, and healthcare data owners). It will include publishing joint work and exploring opportunities to respond to BAA in Darpa, AFRL, and NSF.
1.2 Graphical Illustration

We propose a novel approach for *data provenance, cross-domain software development, global software collaboration, sharing, auditing access and accountability* with enhanced *blockchain-based* [15, 16] mechanisms to build trustworthiness of the software components and ensure identities of the collaborating parties.

Figure 1 illustrates a high-level view of the framework. The idea of blockchain-based data provenance was discussed with Steve Seaberg, Peter Meloy (NGC) and with Vladimiro Sassone (Univ. of Southampton).

![Figure 1. Blockhub: Software Sharing Framework](image)

The core idea is that any two untrusted collaborators, e.g. X and Y, can securely share software in a controlled manner via smart contracts running in the blockchain network that regulates the access to software modules. Every access to the software module is granted or rejected based on the policies established in these smart contracts and every request and transfer of software module is logged as a record in the blockchain’s distributed ledger.

Software modules are stored in encrypted form in the Software Bundle (SB), which is a self-protected structure that contains encrypted software modules, access control policies and policy enforcement engine. Thus, in addition to smart contracts, running in the blockchain network, there are access control policies, embedded into software bundles, that control the access to software modules. In order to serve a request for a software module from other collaborators, access control policies will be evaluated and enforced by the policy enforcement engine. If the evaluation phase is passed successfully, correct decryption keys will be generated on-the-fly to decrypt requested software modules for which the requesting collaborator is authorized.
The proposed solution enables tracking and access auditing for every software module through the following steps:

- **Registration of software attribute and ID information based on software location**: collaborators in the software distribution network will provide Software Bundle ID, linked to the data storage location, to the blockchain’s distributed ledger. Software modules embedded in Software Bundles remain on their respective site in encrypted form. Collaborator will be able to search required software module in the distributed storage using the stored ID information.

- **Smart Contract management**: Each collaborator will be able to manage the smart contract access control policies for its own software module when the ID of Software Bundle is registered in the blockchain. Software owners can decide who can access what particular software module via the distributed ledger using policies in smart contracts.

- **Secure software dissemination through ‘WAXEDPRUNE’ [1, 6] solution**: collaborators will not be able to download software directly from the repository. Software modules are protected and stored in encrypted form using Software Bundle mechanism, built on top of Active Bundle [2, 5]. Each separate software module is encrypted with its own encryption key, generated on-the-fly, based on the execution flow, depending on Software Bundle modules, authentication code and resources: authentication certificate and access control policies. After a client sends a request, accessible software modules will be retrieved from a Software Bundle, decrypted and sent to the requested collaborator based on authentication result, evaluation of access control policies and collaborator’s attributes [1, 5].

- **Automated Process**: The blockchain contains smart contracts that will be triggered when a transaction is requested. These smart contracts will automatically run a series of jobs that will complete the software dissemination process, from verifying the access authorization to starting software transfer in case when authorization has been granted by both smart contract and policy enforcement engine of the Software Bundle.
2 Description of Project

2.1 Statement of Problem

There is a need to design a solution that supports secure cross-domain software development, global software collaboration, sharing, auditing, and accountability, which ensures integrity, trust, immutability, and authenticity of cyber data, user data and software modules. The software exchange model should support role-based and attribute-based access control. Every transaction must be logged and integrity of log files must be ensured. Verification of any transaction any time in the future needs to be supported. In order to ensure integrity of provenance data used to track disseminated software, blockchain technology can be used. Based on the feedback from Robert Pike and NG engineers, a blockchain based solution needs to support the following:

1. Tracking and control of software components that are shared across multiple countries or security domains.
2. Increasing trust for software components developed on sites with low classification level and moved to high-level sides or vice versa (e.g. software developed at ‘Unclassified’ level is transferred to ‘Secret’ level).
3. Automating the export auditing and tracking processes.
4. Cross-domain dissemination of encrypted software modules based on role-based and attribute-based access control.
5. Licensing provenance of deployed software components.
6. Enabling software supply chain that is tamper resistant.
7. Software spillage remediation.
8. Advancing blockchain storage mechanism by using two separate blockchains for significant and insignificant data points.

2.2 State of Current Technology and Competition

Blockchain technology is disrupting the supply chain in the world through its promise of enhanced transparency, security through immutability, and speed. But immutability of a blockchain can be changed in some cases. Instead of having a “mother blockchain”, financial industries have created digital multiple ledgers for different purposes [25]. The initial purpose of the digital ledger is to be open to all, yet some companies are creating “permissioned” blockchains in order to restrict the access to only approved parties.

Compared to regular centralized databases, blockchain provides more trust, robustness and fault tolerance due to its immutability [26]. Blockchain provides disintermediation i.e., instead of relying on a central administrator, the transactions can be verified by the collaborators that can view the digital ledger and provide consensus. However, recent security breaches show that blockchain technology is prone to hacking. The Krypton blockchain, Ethereum-based blockchains, coded in golang, is hacked by “51 crew” [31]. They were able to replicate and manufacture their own blockchain, and push that into production as a real Krypton chain with large amounts of hashing power and DDoS attack on Krypton chain nodes. Thus, in some cases, if the requirements
of a company or a software are filled by regular industry solutions, it is not necessary to use blockchain [27]. It is better to trust the years of development in industry solutions. We will investigate the cost and benefits of blockchain implementation in various scale.

Microsoft has announced that in collaboration with Blockstack Labs, ConsenSys and developers from all over the world, it is working on an open-source, blockchain-based identity system that allows apps and services to interoperate across blockchains, cloud providers and organizations [28]. Recent study [29] demonstrates that trust and immutability are provided through provenance on blockchain technology, where smart contracts can be created. This increases trust and reduces the need for a third party intervention in decentralized systems. Our research will heavily utilize data provenance with blockchain technology.

As for data analytics tools, log management and analysis tools such as Splunk [12], Graylog [13] and Kibana [14] provide capabilities to store, search and analyze big data collected from multiple logs on enterprise systems, allowing organizations to detect security threats and anomalies. Such tools mostly require human intelligence for threat detection and need to be complemented with automated analysis and accurate threat detection capability to quickly respond to possibly malicious activity in the enterprise networks. Splunk imposes significant performance overhead.

Speaking about secure cross-domain sharing of software, a mechanism of micro-policies [21] enforced at a browser's side was proposed to provide confidentiality and integrity of web sessions. The mechanism can be used to ensure secure access to web data by means of the http(s) protocol. Micro-policies are specified in terms of tags, used to label URLs, network connections, cookies; and a transfer function, which monitors security-relevant operations based on these tags. Transfer function defines which operations are permitted by the browser based on the tags. This mechanism is implemented as a Google Chrome extension, Michrome [22].

In our approach, in contrast, the policies are enforced in the Software Bundle, by its policy enforcement engine. The main differences of the suggested micro-policy-based solution from our Software Bundle based solution are as follows:

1. Browser-enforced micro-policies do not provide role based access control [24].
2. Trust level of clients is not constantly monitored and recalculated in the software sharing model
3. The proposed solution records provenance data and provides its integrity by using a blockchain based technique
4. The proposed solution, in addition to access control, supports software spillage detection and remediation

We have discussed access control in blockchain with Steve Seaberg (NG) and he has given us material to read and identified challenges. We will collaborate with him to advance the use of blockchains in federal government and military adoption.
2.3 Proposed Solution and Challenges

We propose the blockchain-based [3] ‘Blockhub’ framework with smart contracts that provides high assurance and highly automated secure cross-domain software development, sharing and audit. Integrity of provenance data is guaranteed by blockchain. Figure 3 provides an overview of the proposed architecture. Blockchain ledgers are used to track and control what software components are shared between entities across multiple security domains. Software components (modules) are stored inside Software Bundles in encrypted form. Software Bundles, described in section 1.2, built on top of Active Bundles [2], ensure that each party can access or update only those software modules for which the party is authorized. Software Bundles contain encrypted software modules, access control policies and policy enforcement engine. When request for a software component reaches the Software Bundle, firstly the requesting client needs to authenticate itself to the Software Bundle (we use certificates for that) and then access control policies as well as client’s attributes, such as level of browser’s cryptographic capabilities [10] and trust level, are evaluated by the Software Bundle kernel. In addition to access control policies, blockchain smart contracts can be used to ensure that two services that don’t trust each other can exchange software modules and update them. Thus, software components developed for entities with high classification level cannot be moved to the low-level classification side. Software components developed at low classification level side have restricted functionality to be executed on high-level classification sides.

All the transactions are recorded in the blockchain and can be verified in the future. Blockchain automates the export auditing and tracking processes. Provenance data as well as Software Bundle kernel are used for software spillage detection and remediation.

Software Bundles ensure tamper-resistance of disseminated software modules. Each separate software module is encrypted with its own encryption key, generated on-the-fly, based on the execution flow, depending on Software Bundle modules, authentication code and resources (certificate and access control policies). Similarly, based on evaluations of access control policies and client attributes, decryption keys for accessible software modules SMi will be derived on-the-fly, based on the execution flow, depending on Software Bundle modules, authentication code, and certificate and access control policies. If some of these parts are corrupted, then an incorrect decryption key will be derived and the software module will not be accessible. Otherwise, accessible software modules will be retrieved from a Software Bundle, decrypted and sent to the requesting party.

Blockchain can be applied when there are multiple untrusted writers and no trusted intermediary is intended. We need to have public distributed ledger of all the transactions that occurred in the system in order to be able to track and control who accessed what software and when. Blockchain application for decentralized software dissemination scenario is illustrated on Fig. 5.

Active Bundles are the core of the ‘WAXEDRPUNE’ prototype [1]. We plan to build the ‘Blockhub’ system on top of this system and deploy blockchain technology and data analytics functionality. The main components of the proposed solution and the challenges involved in their implementation are described below.
2.3.1. Blockchain-based software sharing

Blockchain technology is used to record data provenance [7] and verify any single transaction any time in the future. Blockchain can be viewed as a public immutable ledger for recording sequence of events or history of transactions [3]. This technology is deployed to provide a high-degree of trust, accountability and transparency associated with a set of transactions/events—especially log files and data provenance.

The core idea of blockchain is that each transaction in the public ledger is verified by consensus of a majority of the participating nodes in peer-to-peer network [3]. Once entered, provenance information can never be erased. The blockchain contains a verifiable record of every single transaction ever made in the past in the network.

Each transaction is digitally signed using the private key of the sender and verified with the public key of the receiver. The sender needs to prove her ownership of the private key. Each transaction is represented online as a block, then it is broadcasted to every node in the network and is then recorded in a public ledger. The transactions in one block are considered to have committed at the same time. The blocks are linked to each other in a linear, chronological order with every block containing the hash of the previous block, in the form of a Merkle tree. To solve the problem that there can be multiple blocks created by different network nodes at the same time, each block is accepted in the blockchain only if it contains an answer to a special mathematical puzzle [3], which requires some computational resources at the node. Nodes participating in solving mathematical puzzle are called miners and the first node, to solve the problem, is awarded and it broadcasts the block to the rest of the network.
There is one Merkle tree per software bundle and it gets updated with the hash of the provenance data each time a transaction occurs, i.e. either a software component is read from the Software Bundle or a software component inside the software bundle gets updated by an authorized service. Provenance data is recorded in the blockchain and it contains information on what software component has been accessed/updated, by whom (by which service), when and who sent the Software Bundle to the service. Provenance record may also contain information on the Execution Environment Parameters (e.g. Java version) of the service.

It is important to ensure that data path is known or provable every step of the way [11]. The tracking of data is important for actors (services) involved in data exchange. In trusted environments, applying blockchain technology has less value since data dissemination between multiple trusted writers can be provided by a distributed database [27]. For a database with multiple non-trusting writers the solution still can avoid blockchain and use a trusted intermediary, who is trusted by all the writers, even if they do not fully trust each other [27]. In our scenario, blockchains remove the need for trusted intermediaries by enabling software repositories with multiple non-trusting writers to be modified directly by collaborators. In the software distribution network multiple untrusted collaborators can update software modules, based on collaborators’ authorization level. No central authority is required to verify transactions and authenticate their source. Instead, the definition of a transaction is extended to include a proof of authorization and a proof of validity. Transactions can therefore be independently verified and processed by every node which maintains a copy of the database” [27].

We will investigate the use of lifetime limitations for blockchain. The expiration of the blockchain can be triggered by subjective trigger points such as time, role revocation, etc. Once the blockchain is expired then nobody can write new blocks to that blockchain. A new blockchain will need to be replicated from the expired one with updated access control permissions. The expired blockchain can still be used for verification purposes as a legacy.
2.3.2. Decentralized blockchain-based software development

As a use case scenario, we propose privacy-preserving blockchain-based cross-domain dissemination of software modules in untrusted environments. Fig. 5 illustrates this scenario. Software modules in the decentralized network are sent between services in the form of Software Bundles, built on top of Active Bundles [5]. As an initial step, software is created by the owner, which is NG in the United States and is stored in the form of a Software Bundle. A hash of software is produced and stored in the blockchain. Next the software owner (NG US) sends software (set of software modules) to NG in UK. NG-UK verifies the origin of the software and then can read/update it, based on access control policies. Every time a software module is updated by a collaborator (e.g. NG in UK adds new functionality to software module), this needs to be approved by a consensus of other nodes, miners. For instance, by a consensus of several NG offices. Once a transaction is approved by a consensus of other participants, the provenance record is generated. The read/write provenance entry has information on what software module got accessed/updated, when, by whom and who sent it. Next the hash value of software is computed and added together with the provenance record to blockchain. It is infeasible for an attacker to modify the provenance record of a transaction since it has “not only to generate a block by solving a mathematical puzzle, but it also has to race mathematically against the good nodes to generate all subsequent blocks in order for it to make the other nodes in the network accept its transaction and block as the valid one” [3]. This becomes even more difficult since blocks in the blockchain are cryptographically linked together and stored in the form of a Merkle tree of hash values, so that it is impossible to corrupt the hash value of one node of a tree without modifying the hash values of other nodes. In our scenario, NG in UK cannot repudiate the fact that it accessed and updated a given software module on a particular date. A collaborator cannot corrupt its log file since it will make the whole blockchain invalid and other nodes of the network will recognize it.
The blockchain protocol repeats for the other message, when NG-UK propagates the software to other nodes in the network, e.g. to NG in Australia. As a final transaction, NG in Australia sends software to some other collaborator, which might be allowed to access only a certain restricted set of software modules.

2.3.3. Efficient blockchain storage
We will utilize customized direct acyclic graph (cDAG) data structure for blockchain implementation. A DAG has no direct cycles and it is a finite directed graph and it aids autonomy by reducing the interventions from external sources and making it easier to create robust blockchains protocols. Merkle tree optimizations will be implemented by assigning quantitative measures to the provenance data points to decide the significance of each data point so that they are sufficient for data analysis. There will be two separate blockchains for significant and insignificant data points. This will reduce the size of Merkle tree and number of blocks. Extensive performance measurement experiments will be run to estimate cost/benefits of the proposed solution.

2.3.4. Challenges of blockchain technology deployment
1. Performance
Blockchain is replicated to all the network participants and this imposes a high performance overhead.

2. Access Control (Read)
In case of access revocation or subject’s role change, access to data or to software must be revoked immediately within an information system when authorization is no longer valid. However, revoked access to data on a blockchain can be bypassed in the following ways:
(2a) by replaying old blocks against an empty blockchain and stopping before the revocation block is appended;
(2b) An attacker holding a copy of a blockchain could use a modified client to just ignore the revocation block.

Even if read access to a local blockchain requires an off-chain token handshake with a centralized authority for authorization; then that token would continue to work forever in the future. The requirement to revoke previously granted access can be bypassed by rolling the local clock back and restoring unauthorized access to blockchain data.

3. Software Spillage Remediation
Software spillage occurs when a software module from a higher classified level is spilled to a lower classified level. Software spillage is remedied by securely deleting the software on the lower classified system. Revoking read access to the spilled data on the blockchain might be considered as a solution, but it requires solving the Access Control (Read) issue above.

The challenges and tasks for blockchain are based on feedback and advice from Steve Seaberg (NG). We plan to collaborate with Steve Seaberg, Peter Meloy, Jason Kobes, Donald Steiner and other engineers at NG and Vladimiro Sassone (NGCRC-international partner and University of Southampton, UK) to work on blockchain-based methodology for tracking and control of software components that are shared between entities across multiple security domains.
2.3.5. Software Spillage Remediation

In SOA, the services, including ones from untrusted environments, can interact and share software modules with each other. Our approach ensures that service can access only those software modules for which the service is authorized. Software owner availability is not required to provide access control policies enforcement. However, authorized services may spill software to unauthorized ones. Based on the form of spillage, the protection mechanism is provided by an Software Bundle [1, 2], web crawler and digital watermark. Log files having records on what software modules were accessed/updated by whom and when are used to do forensics and investigate spillage incidents and identify spilling service. Integrity of log files is provided by blockchain technology, described above. Either the whole software (Software Bundle) or separate software module can get spilled.

![Diagram](image)

*Figure 6. Software module SM1 spillage from Service X to Service Y*

Let us denote software \( S = \{SM_1, SM_2, \ldots, SM_n\} \); set of policies \( P = \{p_1, p_2, \ldots, p_k\} \). Software Bundle contains encrypted software modules

\[
Enc[Software(S)] = \{Enc_{k1}(SM_1), \ldots, Enc_{kn}(SM_n)\}
\]

Access control policies embedded into Software Bundle are enforced by the policy enforcement engine, which is part of the Software Bundle. Let’s say service X receives Software Bundle from the Service A and X is authorized to get software module SM1 from the Software Bundle and no other software modules. We address two spillage scenarios.

1. X leaks the whole Software Bundle to Y. Service Y won’t be able to decrypt software modules it is not authorized for. When a service tries to access the software module from the Software Bundle, which is implemented as a JAR-file, the Software Bundle’s kernel notifies the Central Monitor about what type (class) of software module the service attempts to decrypt, when and who sent the Software Bundle to this service. The unsuccessful attempt made by unauthorized service to access software module it is not authorized for, will be detected by Central Monitor (CM) and software spillage alert will be raised and recorded. CM will check the transaction notification sent by an Software Bundle and detect that SM1 is not supposed to be at Y. Until the acknowledgment is received from the CM, decryption process will not start. It prevents intentional isolation of malicious node Y in order not to let notification message being sent from Y to CM.
Unclassified

(2) X leaks separate decrypted software module SM$_1$ to Y. Now protection provided by Software Bundle’s policy enforcement engine is gone since SM$_1$ has already been extracted from Software Bundle by authorized service X and decrypted. In this case, spillage protection mechanism relies on digital watermarks [23], that can be verified by CM or by web crawlers. If attacker uploads the illegal copy of software to publicly available web hosting that can be scanned by web crawlers, then web crawler can verify the watermark and can detect copyright violation. We can force the software module to contact CM each time the software module starts execution. Then CM can check the digital watermark of the software module and terminate the software execution if digital watermark does not match.

Other spillage remediation methods may involve giving less information (or statistical or incomplete or less sensitive information or with added noise) at first. E.g. it could be software module with restricted functionality. Then the system observes whether the recipient is satisfying obligation before the complete (whole) sensitive software module is given. The key challenge here is that it is hard to come up with an approach that covers all possible cases of software spillage.

Damage assessment model for software spillage incidents will be developed. After software spillage is detected damage is assessed based on:

- To whom was the software spilled (service with low trust level vs. service with high level of trust)
- Sensitivity (Classification) of spilled software (classified vs. unclassified)
- When was software spilled (new vs. obsolete version)

\[
\text{Damage} = P (\text{Software is Sensitive}) \times P (\text{Service is Malicious}) \times P(t)
\]  

(1),

where P(t) is the probability function for data sensitivity in time.

Fig. 7. Software sensitivity probability functions
Fig. 7 shows three different probability functions for software sensitivity. Software-related event (e.g. new release) occurs at time $t_0$. Threat from software being spilled before $t_0$ is high. Threat from software being spilled after $t_0$: either goes to zero right away (e.g. public release) or degrades linearly or remains to be high with time.

### 2.3.6. Data Analytics for Software Spillage Detection

Analytics over provenance data will be used in our framework to identify threats, detect software spillages and generate recommendations on system’s configuration. Performing analytics on provenance data that is obtained through blockchains will enable to evaluate the trust level and quality of shared software based on parameters such as origin of software, trustworthiness of nodes generating and sharing software, and integrity of software. Trust level of services is constantly recalculated by a central monitor, based on several metrics including number of sent/received software requests, CPU and Memory usage, number of connection failures, etc. As blockchain permits the participation of untrusted parties in the network, malicious behavior can be detected through the analysis of the transactions issued by an entity. Detection of abnormal transactions enables the system to update trust values of participating nodes by determining the timing of leak and the amount of spillage. Services can continuously generate massive amounts of data that need to be processed and analyzed in real-time or near real-time to detect anomalies. Open-source technologies such as Apache Kafka [8] is an industry standard used for ingestion and queuing; and Apache Spark [9] is an industry standard used for streaming processing and large-scale data analytics. The proposed solution will include investigation of Kafka and Spark modules for large-scale data analytics.

Our approach ensures that collaborators can access only those software modules for which they are authorized. When a collaborator tries to access the software module from the Software Bundle, the Software Bundle’s kernel notifies the Central Monitor about what type (class) of software module the service attempts to decrypt, who the Sender is and when the access attempt happens. These parameters of the transaction are recorded in the central log file. Microsoft SQL Server Data Services (SSDS) [17] can be employed for that purpose. Based on obligations (access control policies), central monitor either approves or denies the decryption. Log records are then extracted from the central log file, transformed and loaded into a relational database. Integrity of the log records is ensured by applying blockchain technology. Data extraction, transformation and loading can be done by means of Microsoft SQL Server Integration Services (SSIS) [18]. After that analytical methods from Microsoft SQL Server Analysis Services (SSAS) [19] can be applied to process the provenance data in order to:

1) Categorize services into healthy and suspicious ones, involved in software spillages
2) Compute and visualize on dashboards the statistics for services exchanging software modules in SOA, including the following parameters:
   a. Number of accepted/denied requests for software modules
   b. Number of Software Bundles sent to other known nodes
   c. Number of Software Bundles sent to nodes from untrusted environments

To visualize the results and generate reports, Microsoft SQL Server Reporting Services (SSRS) [20]
can be used. Online log analysis can be also implemented by using the set of Elasticsearch [15], Logstash [16] and Kibana [14] tools (ELK). Central log file that is maintained at a central monitor, can be used as an input in CSV-format for ELK tools.

In addition to analytics over provenance data (what software module got accessed/updated by whom and when, who sent the software module), analytics can be built over the software modules stored in the Software Bundle in encrypted form. For instance, a role ‘Data Analyst’ can be created which can access descriptions of software modules stored in encrypted form in the Software Bundles together with software modules.

![Fig. 8. BI model for provenance data analysis](image)

### 2.4 Distinctive Attributes, Advantages, and Discriminators

**Privacy-preserving cross-domain software sharing:**

It will provide that any untrusted collaborators can securely share software in a controlled manner across multiple security domains via smart contracts running in the blockchain network that regulates the access to software modules. Every access to the software module is granted or denied based on the policies established in these smart contracts and every request and transfer of software module is logged in the blockchain’s distributed ledger. Software modules are stored in encrypted form in the Software Bundles, which, in addition to smart contracts, enforce their own access control policies. Software Bundles provide tamper-resistance for software supply chain and support attribute-based context-aware software sharing, in addition to role-based access control.

**Software access auditing and tracking:**

Integrity of provenance data is provided by blockchain technology. Every request for software is logged in the blockchain’s distributed ledger. Any transaction, i.e. any request to read/write software module, can be verified any time the future and cannot be corrupted without being detected by other software collaborators. Service (software collaborator) cannot repudiate any transaction that it issued. Cost analysis for provenance recording will be conducted. We will consult with Peter Meloy (NG-UK) on performance improvement techniques.

**Software spillage detection:**

It will be provided by Software Bundles, web crawler and digital watermarks. Each time before decrypting requested software module, stored in Software Bundle in encrypted form, the Central
Monitor will be notified about who (what service) tries to decrypt what software, when and where did the software arrive from. Permission to decrypt will be granted or denied based on the evaluation against database of obligations at Central Monitor. Each time when the software module starts execution, it will have to contact Central Monitor that will verify the digital watermark of the software module. Web crawler can also verify digital watermarks on publicly available network nodes. Thus, spillage of software from nodes with high classification level to low classification level will be detected.

Data Analytics:
Data analytics over provenance data will be used to identify anomalies, detect software spillages and generate recommendations on system’s configuration. Several machine learning algorithms (e.g. SVM-based, K-means clustering) can be run in real-time or near real-time to process provenance data obtained from blockchain. The statistics on software attributes and requests will be visualized on a dashboard.

2.5 Tangible Assets to be created by Project

2.5.1 Software

- ‘Blockhub’ prototype: it will be built on top of the “WAXEDPRUNE” prototype [1], its source code is available at github [30]. ‘Blockhub’ will support blockchain technology with smart contracts, which ensures integrity of log files and provenance records, as well as providing secure and trackable software sharing between untrusted collaborators (entities).
- Central Monitor: it will be implemented as an extension required for software spillage detection. Central Monitor will run as a service on a trusted site. Central Monitor will log and analyze all service interactions in order to detect whether the software module arrived to a particular entity (collaborator) is supposed to be there, according to obligation policies.

2.5.2 Documentation

We will provide four types of documentations that would help NGC researchers. They include:
1. **Source code:** Code for the software will be well self-documented (including comments) for possible extensions/modifications by future developers.

2. **Deployment and user manuals:** All software components created in the project will be clearly documented with deployment guides and user guides on how to use each component separately as well as how to use the whole prototype.

3. **Reports:** We will provide mid-term and final reports that describe algorithm implementations, and the experimental results that characterize the performance of the presented solutions. These results will include both system performance and security evaluation of the system.
4. **Demonstrations:** To be made at NGC meetings, to NGC researchers, and at Tech Expo 2018.

We will provide high-quality documents adhering to the standards used at NGC. Code for the software will be commented clearly for possible extensions/modifications by future developers.

## 2.6 Technical Merit and Differentiation

The proposed approach offers several advantages for privacy and integrity-preserving cross-domain software dissemination and sharing in collaborative untrusted environments. The main benefits of the proposed approach are:

1. Tracking and control of software components that are shared between entities (collaborators) across multiple countries or security domains by providing integrity of provenance records.
2. Increasing trust for software components developed on sides with low classification level and moved to high-level sides or vice versa.
3. Automating the export auditing and tracking processes.
4. Cross-domain dissemination of encrypted software components that supports role-based and Attribute based access control. The software sharing model considers the following attributes of a subject (service):
   a. Level of browser’s cryptographic capabilities (check how secure the client’s browser is) [10].
   b. Trust level, which is constantly recalculated by a central monitor.

The software sharing model has the following distinctive features:

a. Software owner’s availability is not required to enforce access control policies since access control policies are attached together with software (stored in encrypted form) and policy enforcement engine
b. Complicated access control policies are supported
c. Agnostic to policy specification language
d. Software updates are supported for services
e. Novel secure scheme of encryption/decryption key generation. Keys to encrypt/decrypt data items stored in Software Bundle, are generated on-the-fly, based on the execution flow [5]. Keys are not stored neither inside Software Bundle nor on some Trusted Third Party
f. Supports context-aware data dissemination (e.g. normal context vs. emergency)

5. Licensing provenance of deployed software components.
6. Enabling software supply chain that is tamper-resistant.
7. Trust-based software dissemination.
8. Software spillage remediation.
9. Advanced blockchain storage mechanism that uses two separate blockchains for significant and insignificant data points
10. Real-time data analytics over provenance data to detect anomalies, make predictions and generate recommendations.
3 Project Milestones

3.1 Statement of Work

The following tasks will be completed in the ‘Blockhub’ project:

1) Development of robust mechanism to store provenance data used for software spillage detection. Provenance data compression techniques will be investigated.
2) Using blockchain technology with smart contracts for software sharing between two untrusted entities (collaborators). Investigation of time limitations for blockchain.
3) Deployment of cross-domain dissemination of encrypted software modules that supports role-based and attribute-based access control.
4) Automation of access auditing and tracking processes.
5) Development of tamper-resistant software supply chain.
6) Development of advanced blockchain storage with two separate blockchains for significant and insignificant data points.
7) Development of schemes to control software spillage.
8) Investigation of time limitations for blockchain when a new blockchain needs to be replicated from the expired one with updated access control permissions.
9) Running real-time data analytics tools over provenance data to detect anomalies, make predictions and generate recommendations.
10) Presentation in NGCRC symposium in 2018 and submit final report in August 2018
11) Possible demonstration at NGC TechFest 2018 and integration in NGC prototypes demo

The following experiments are proposed for proof of concept and fine tuning of proposed solutions.

Experiment 1. Evaluation of blockchain-based integrity scheme by corrupting the transaction record in the log file

- **Purpose:**
  - Identify network participant with the corrupted log file
  - Make sure that suggested blockchain-based provenance scheme provides integrity of log files

- **Input:**
  - Software modules stored in the form of Software Bundles
  - Access Control Policies
  - 3 services sharing software

- **Output parameters:**
  - Data Provenance Log (integrity ensured by blockchain)
  - Blockchain containing records on read/write transactions, i.e. read/update Software Bundle operations
Experimental Setup:
- Three Node.JS services (Service 1, 2, 3) updating and sharing software modules
- Software Bundle containing software S, that consists of access control policies P and N software modules
- All the read/write transactions are verified by consensus of a majority of the network participants and stored in the blockchain
- Service 1 authorized to read/write software module 1 reads/updates it
- Service 1 sends it to Service 2 who is not authorized to access Software module 1
- Service 2 tries to access software module 1. Access is denied, software spillage alert is raised
- Service 1 tries to modify (corrupt) its log file of transaction trying to hide the fact that it sent software module 1 to unauthorized service 2

Experiment 2: Evaluation of software spillage remediation
- Purpose:
  - Identify details of what software, to whom, from where and when got spilled
  - Isolate software spillage
  - Identify vulnerabilities in the system
  - Assess software spillage damage
- Input:
  - Software modules stored in the form of Software Bundles
  - Access Control Policies
  - 3 services exchanging data
- Output parameters:
  - Data provenance log (integrity ensured by blockchain)
  - Sensitivity of spilled software
  - Software spillage vulnerability
  - Timing of spillage
  - Extent of Damage
- Experimental Setup (see Fig.6):
  - Three Node.JS services (Service A, X, Y) updating and sharing software modules
  - Software Bundle containing software S, that consists of access control policies P and N software modules
  - All the read/write transactions are verified by consensus of a majority of the network participants and stored in the blockchain
  - Central Monitor with database of obligations (policies)
  - Service A authorized to read/write software module 1 reads/updates it
  - Service A sends it to Service X who is also authorized to access software module 1
  - Service X successfully accesses software module 1
Service X sends the whole Software Bundle to Service Y. Service Y tries to access software module 1. Access is denied, software spillage alert is raised and recorded.

We will conduct cost analysis of blockchain-based provenance recording and extend our approach for adaptive provenance recording decisions. We will explore case-driven scenarios for compression of provenance data.

3.1.1. Integration with NGCRC and NGC IR&D Projects

We aim to collaborate with and contribute to the following IRADs, which are closely related to our work:

1. Northrop Grumman Mission Systems - Engineering Center of Excellence (eCOE)'s “Cyber Resilient DevOps” (Steve Seaberg).
2. Multi-intelligence (MINT) Enterprise Analytics for dealing with semantics of provenance data (Brock Bose) and EIM&Analytics to transform data into actionable insights.
3. Cross domain solutions protecting critical information and technology.

We plan to collaborate with Steve Seaberg, Peter Meloy, Jason Kobes, Donald Steiner, Brock Bose (NG) and Vladimiro Sassone (University of Southampton, UK) to work on the proposed Blockhub.

We plan to coordinate with Jason Clark/Joshua Bernstein to target BAA in Air Force Research Lab, Rome, NY. We like to apply the proposed research to medical data in collaboration with Dr Sam Shekar and Dr. Roderick Son in Health division.

We plan to prepare a demonstration of the blockchain-based privacy-preserving cross-domain software development and sharing for the NG Tech Expo in 2018. We plan to integrate the knowledge from this research to advance the efforts of other IRADS.
3.2 Milestones and Accomplishments
The following Table shows the list of tasks to be accomplished during the project period, broken down in a quarterly basis. We plan to hold weekly meetings in Spring 2018 with NGC researchers to accomplish the development of demos for Tech Expo 2018.

<table>
<thead>
<tr>
<th>Task</th>
<th>Q1 (Sep - Nov)</th>
<th>Q2 (Dec - Feb)</th>
<th>Q3 (Mar - May)</th>
<th>Q4 (Jun - Aug)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop robust mechanism to store provenance data</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop blockchain-based software sharing with smart contracts</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Develop cross-domain sharing of encrypted software modules</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop tamper-resistant software supply chain</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Develop software spillage control</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automate software export auditing and tracking processes</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Develop advanced blockchain storage</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Investigate time limitations for blockchain</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Develop real-time data analytics over provenance data to detect data anomalies</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Prototype demonstration at NGC Tech Expo 2018 and Reports</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1: Milestones and Accomplishments
4 Project Budget Estimate

The project will involve one faculty, one Ph.D. student (working on Ph.D. dissertations on software spillage, provenance and analytics) and one postdoc/programmer (who is continuing work on the current NGCRC project and will facilitate demos at Tech Expo). Denis Ulybyshev has worked on past NGCRC funded research towards his Ph.D and has received an award for one year research assistantship (worth $45K) from corporate partners of computer science department for one year. His labor and salary is not charged to NGCRC. Budget will consist of salary for the faculty, salary for the programmer/postdoc, and salary for Ph.D. student. The total budget including fringe benefits and Purdue University overhead will be $149,999.

<table>
<thead>
<tr>
<th>Category</th>
<th>Items</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labor Hours</td>
<td>Total Hours: 3120</td>
<td>84,907.00</td>
</tr>
<tr>
<td>Materials and Equipment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Travel</td>
<td></td>
<td>5,246.00</td>
</tr>
<tr>
<td>ODCs</td>
<td>Tuition Fees</td>
<td>10,262.00</td>
</tr>
<tr>
<td>Other</td>
<td>Indirect costs @ 55% MTDC</td>
<td>49,584.00</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$149,999.00</td>
</tr>
</tbody>
</table>

Table 2: Project Budget Estimate

References
Unclassified


