On the (f)utility of untrusted data sanitization

Ashish Gehani David Hanz John Rushby Grit Denker Rance DeLong
SRI International, Menlo Park, California, USA

***Abstract*—Data sanitization has been studied in the context of architectures for high assurance systems, language-based information flow controls, and privacy-preserving data publication. A range of sanitization strategies has been developed to address the wide variety of data content and contexts that arise in practice. It is therefore tempting to separate the complex downgrading operations into untrusted data sanitizers while leaving the verification of security policy to simpler trusted guards that mediate information flow between different sensitivity levels. We argue that this can be a false economy and may result in more restrictive information flow than is necessary. We also observe that the guarantees provided by language-based declassification algorithms do not hold without exacting requirements for the runtime environment, and that the satisfaction of these requirements is the precise goal of MILS architectures, making the two disciplines well-matched complements.**

I. Introduction

Cross-domain solutions (CDS) for assured information sharing (AIS) are an integral component of the U.S. Defense Department's global information grid (GIG) [16]. (Data diodes, filters, sanitizers, and downgraders are examples of CDS devices.) The information assurance architecture documentation of the GIG states that previous CDS approaches are inadequate, particularly for network-centric operations [6].

The months (or even years) necessary to complete the approval process for legacy CDS deployments must be replaced by more agile mechanisms. The level of agility envisioned can be gauged by the U.S. Navy's requirement of a maximum setup time of one month (from event announcement to execution) for a U.S.-coalition training event, with the requirement dropping to one week for a training event composed of only U.S. Government entities, and to one day for an event involving only the U.S. Department of Defense [43].

The broad approach for realizing the CDS vision involves decomposing the problem into a set of smaller subproblems that are more tractable and then integrating the component solutions together to achieve the desired capability. The building blocks include formal specifications for generic downgrading filter engines, formal languages for specifying data sanitization rules [13], downgraders for specific data types, and attribute-based access control [26].

The use of precertified commercial off-the-shelf (COTS) CDS can facilitate rapid deployment in the field. However, the availability of COTS devices remains dependent on their timely evaluation. We examine approaches that advocate effecting the complex data sanitization operations in untrusted monolithic versus trusted but decomposed sanitizers while limiting the role of trusted guards to verifying that the data has been downgraded, thereby speeding up CDS evaluation time.

Section II describes the diverse settings in which the need for data downgrading has arisen. Section III explains why the increasing complexity of downgrading operations poses a challenge to certifying the available solutions, the motivation for a paradigm shift in how downgrading is performed, and the limitations of current data sanitization technology in the new framework. Section IV describes an architecture for decomposing downgrading to facilitate certification, and composing the assurance provided. We conclude in Section V that despite the challenges, it is possible to deploy data sanitization technologies for the increasingly complex data and contexts of high assurance systems within the target timeframes.

This material is based upon work sponsored by the Air Force Research Laboratory (AFRL) through the Air Force Cryptographic Modernization Program Office under Contract Number FA8750-10-C-0230. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL.

II. Data downgrading

We examine the use of data downgrading in three settings – high assurance systems, language-based security for mobile code, and privacy-preserving data publication. In each case we describe the context in which the need for data downgrading arose and the role it plays in an information system's architecture. We distinguish between the terms *sanitizer*, *guard*, and *downgrader*, with the first performing a reduction in data fidelity, the second verifying this, and the third effecting the combination.

A. High assurance systems

High assurance systems operate in environments where the repercussions of security breaches are very significant. Examples of this are aircraft navigation, where a fault could lead to a crash, critical infrastructure control systems, where an error could cause toxic waste to leak, and weapons targeting, where an inaccuracy could result in severe collateral damage. In such operational environments, the impact is virtually irreversible and must therefore be prevented even if it is likely to occur with low probability.

The most conservative approach to preventing data leakage is to ensure that there are no channels through which information can flow between components that have differing security classifications. However, this prevents legitimate transfers from lower security sources to higher security destinations. Instead, unidirectional communication links with data diodes can be used to facilitate legitimate flows from a lower to a higher security classification level.

In the early 1990s, the Australian Defence Science and Technology Organization (DSTO) funded the development and commercialization of such a device [38]. The U.S. Naval Research Laboratory developed a downgrader with a reverse channel that allows acknowledgments to be returned, enabling reliable communication that is otherwise not possible [23]. The extra functionality is challenging to design since the acknowledgments introduce the possibility of a covert channel if the endpoints have been compromised.

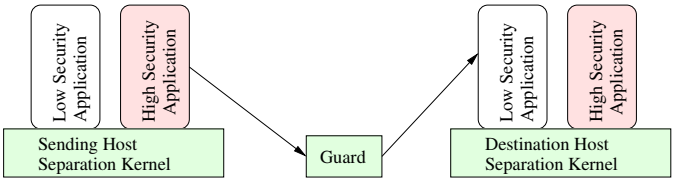

Fig. 1. A *guard* intercedes on channels between hosts to filter data flowing from a higher security level to a lower level.

If a system's security-critical components are decomposed into modules that can each be completely verified and between which no unauthorized interaction can occur, then the integrated system's security can be assured. Rushby introduced this approach with the use of a *separation kernel* [32] that isolates partitions running with multiple independent levels of security on a single host. Information can flow between partitions only through unidirectional communication and control channels. Each such channel can have a *guard* interposed to enforce security properties by filtering the data flowing through it, as illustrated in Figure 1.

The advent of the Web led to the need for downgraders that could handle a wide range of data [31]. The devices would be able to operate on information, transforming it to a form suitable for release, or sanitize the information by redacting portions of it. If neither option proved sufficient, the information flow could be completely blocked.

B. Language-based information flow control

By the late 1990s, execution of untrusted mobile code (such as Java [19] applets) was widespread. The access control [46] and sandboxing [18] techniques used to protect the host systems prevented legitimate interactions between mobile and host code, motivating the need for more flexible information flow controls [33]. Simultaneously, programming languages were augmented with security types [44] that enabled the enforcement of Goguen and Meseguer's *non-interference* [17].

Subsequently, Myers and Liskov introduced the *decentralized label model* [30], describing how labels could be applied to a programming language and then used to check information flow policy compliance in distributed systems. The framework includes a *declassify* function for downgrading data if the owners' policies so allow, as illustrated in Figure 2. The model allows principals to define their own downgrading policies.

Over the past decade, numerous approaches have been proposed for language-based declassification [35]. The first category focuses on *who* is performing the downgrading – for

```
boolean authenticate(String user,
                     String password){
    boolean match = false;
    String storedHash =
            lookupPasswordDatabase(user);
    String passwordHash =
            computeHash(password);
    if(storedHash.equals(passwordHash)){
            match = true;
    }
    declassify(match);
    return match;
}
```

Fig. 2. Language-based security can track the sensitivity of data through the type system with downgrading used to reduce the security levels of variables. In the example above, since the security type of the `password` variable is high, the output of calling the *computeHash()* method on it must also be stored in a high variable – that is, `passwordHash`. Similarly, the implicit flow [10] possible through the *equals()* method requires that the `match` variable must also be high. It is therefore necessary to downgrade the `match` variable to allow it to be returned to a calling context with a low security level. This is accomplished explicitly through the *declassify* operation.

example, in the decentralized label model this is the owner of the data being declassified [30]. The next category describes *what* is being downgraded, such as expressions marked with *escape hatches* [34]. The third category of declassification identifies *where* in the code the downgrading occurs – for example, at points in the code where cryptographic functions are used [20] or where the information flow security policy is updated [21]. The final category covers *when* the downgrading occurs, encompassing a range of temporal aspects, from the time-complexity before data release [45] to the conditions that must hold before declassification [8].

C. Privacy-preserving data publication

The third setting where data sanitization has been extensively studied is the publication of privacy-sensitive data. Individuals contribute their records to a trusted publisher that processes the collected information. The results are stored in a database that can be queried [15]. To preserve the privacy of the individual record owners, a downgrader sanitizes information derived from such databases, as illustrated in Figure 3.

In early work, Adams proposed perturbing the query inputs and outputs, and restricting the number of queries [1]. Since then a range of strategies has been utilized [7], including *generalization* that coarsens, abstracts, or collects multiple data in equivalence classes, *suppression* that removes records from the sanitized output, *swapping* that interchanges attributes from different records, *randomization* that adds noise to perturb the data, and *multi-views* that provide sanitization through diverse perspectives.

The approaches may be domain agnostic and focus on *quasi-identifiers*, data fields that are potentially sensitive. Examples of this include *k-anonymity* [39], which aims to ensure

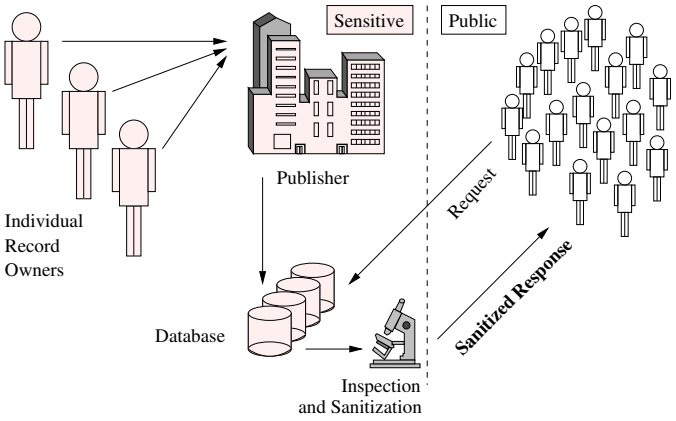

Fig. 3. A *downgrader* sanitizes sensitive data before publication.

that the output contains at least k records with the same quasi-identifiers, *l-diversity* [28], which ensures that auxiliary fields contain at least l different values, and ϵ*-privacy* [29]. Alternatively, the approaches may be customized to specific data types, such as network addresses [47], data formats, such as audit logs [25], or specific application domains, as is the case for each scheme that guarantees *differential privacy* [12].

III. Certification challenge

A. Complexity costs

Since downgraders operate at the boundary between data of different sensitivity levels, they are security-critical components of a computing infrastructure. Assurance of the security of a system is therefore dependent on being able to verify the correctness of downgrader operations.

Early downgrading focused on specific data types and predefined contexts. For example, the U.S. Department of Defense's Global Positioning System used to downgrade the location information available to civilians (and therefore also adversaries) by adding a pseudorandom error to part of the signal [24]. While this approach simplified the process of verifying that the downgrading operation conformed to its specification, the need for more complex downgrading policies became apparent with the development of *differential GPS*. (The errors added during downgrading could be calculated in real time by a receiver at a known location that then retransmitted the error stream to consumers who could use it to determine their own location with greater accuracy.)

As the range of data content, environments from which it originates, and kinds of operations being performed on it continues to increase, so has the complexity of the rules used to downgrade data flowing between different security classifications. The use of language-based information security depends on the correctness of the declassification policies, language compiler, and implementation of the security type system in the runtime. Similarly, the statistical guarantees provided by privacy-preserving data publication algorithms depend on the soundness of the data sanitization infrastructure.

Certification of a downgrader requires detailed requirements and specifications, proofs of correctness, a structured design process, detailed documentation, and the development of test cases with sufficient coverage. Thus, as the downgrader becomes increasingly complicated, the cost of formal assurance and the time that elapses before it can be deployed in the field both grow rapidly.

B. Blocking guards

In 2005, Brian Snow (who was then Technical Director of the U.S. National Security Agency) alluded to a strategy to address the growing economic and temporal cost of certification. He stated that "we need more focus on how to use safely security gear of unknown quality (or of uncertain provenance)" [37]. This would be realized in the context of downgrading by separating the data sanitization into an untrusted component that did not need certification while leaving only the verification of the data security policy to the trusted guard, as illustrated in Figure 4.

We refer to the above division of functionality as the *blocking guard* paradigm. In particular, such guards would be able to perform only two actions – they could either examine a piece of data and deem that its passage conforms to the information flow policy in effect, or they could block the data and prevent it from passing through. In principle, the limited number of actions that the guard could take would reduce its complexity and therefore the certification cost and time.

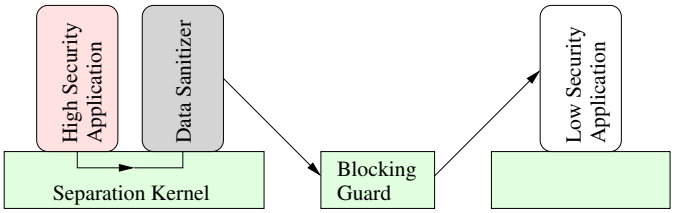

Fig. 4. Data sanitization is separated here from verification, which is done in the trusted guard. However, this approach works with an untrusted sanitizer only if it can provide a *witness* to attest the correctness of the sanitization.

C. Implications of the blocking guard paradigm

In practice, restricting a guard's actions has consequences. If the trusted guard were to perform the data sanitization operations, no further assurance would be needed that the output was downgraded. However, if a sanitization operation was instead performed externally, the guard would need to receive a *witness* that attested the fact. In the absence of a witness, an untrusted sanitizer could perform less (or even more) sanitization than it claimed to have done, and introduce a covert channel past the guard by encoding information in the sanitization process. For example, a stream of zeros and ones could be encoded by a sanitizer that always either redacted one or two adjacent words, depending on the value it was encoding.

At one end of the range of possible ways the communication could occur, the sanitizer could use an algorithm that generated a witness that could be verified in conjunction with only the downgraded data. The guard would require no access to the original data itself. For a range of sanitization operations, no

witness can be generated. In such cases, parts of the data would need to be forwarded to the guard, which must then either be able to perform the sanitization operations (which is counter to the goal of simplifying the guard) or block the data.

Commercial data sanitization systems, such as Raytheon's High Speed Guard [22] and General Dynamics' Tactical Cross Domain Solution [41], do not emit witnesses of the form described above. (Our conclusion is based on the limited documentation available [11] for them.) In the context of language-based information flow security, certificate-based declassification has been proposed [42]. However, this utilizes a public-key infrastructure and authorization policy to justify when the sanitizer can perform declassification rather than providing the guard with a proof of its occurrence. Finally, though cryptographic algorithms have been proposed for privacy-preserving data collection [48], these do not provide a sanitization witness. Instead, privacy-preserving data publishing algorithms trust the data aggregator [15].

The blocking guard paradigm can provide utility when the sanitization process must explore a large state space to satisfy a set of downgrading constraints. In this case, the sanitizer can communicate the selected operations and parameters to the guard as part of the witness, and the guard will be able to efficiently verify the sanitization without repeating the entire search. While such policy-based downgrading has been proposed [9], current solutions do not employ it.

IV. Decomposing sanitization

Previous approaches for handling data in multi-level secure systems have relied on separate instances of an application running at each security level [14], [40]. In contrast, we address the problem of downgrading data that has components with multiple classification levels, as occurs during complex joint training missions [5]. We advocate an approach that leverages the nature of the data being downgraded and the available trusted computing infrastructure to decompose the downgrading functionality to the point that each module can economically be formally specified and have its operational behavior verified. This is illustrated in Figure 5.

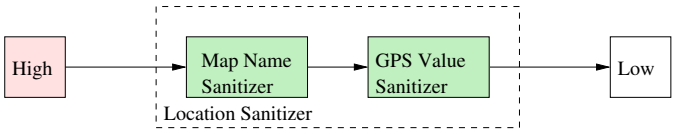

Fig. 5. Data sanitization functionality can be decomposed until each sanitization primitive can be certified. For example, sanitization of location information can be split into one operation that redacts blacklisted names from a map while a second perturbs the numeric values in GPS values.

A. Architectural context

Since we use an architecture-based solution to make complex data sanitization practical, we first describe the MILS [5] context in which it is framed.

The MILS philosophy advocates a top-down approach to securing a system, mirroring the architectural pattern employed by the *multiple independent levels of security* initiative [3]. The essence of the MILS architectural pattern is to first describe the required security policy in terms of a *policy architecture* (typically with a diagram in which boxes encapsulate processing functions and arrows depict information flow). The difficulty of making the assurance case is then examined, assuming a direct mapping between the architecture and a physical realization of it. If there is difficulty in proving the assurance case, then the policy architecture is further decomposed until a point is reached where the assurance case can be proved.

The MILS approach separates the desired security properties from the resource-sharing problem. Commodity implementation of individual modules is facilitated through the development of *protection profiles* that articulate the properties that the resources must possess. Downgraders form one class of such functionality. A significant reason that current downgraders are limited to simple operations, such as redacting patterns contained in a database of blacklisted terms or perturbing numeric values using fixed rules [36], is to ensure that they are consistent with the MILS philosophy of individual modules implementing well-defined security properties.

B. Sanitizer selection

Downgraders operate on structured data that has a defined data model [4]. The information is transferred in fixed format messages with multiple data fields and the allowable range for each specified in the associated metadata. The information is intended to support interoperability between machines. Consequently, any piece of data d_i that is a candidate for sanitization has an associated set $M(d_i)$ of constituent objects used in its data model.

If we denote the type of data d_i by $\tau(d_i)$, then $S(\tau(d_i))$ represents a trusted sanitizer that can operate on data d_i. A downgrader is trusted if it has been certified. If no such sanitizer is available for d_i, then sanitizers must be found for the set $T(d_i) = \{\tau(d_j) : d_j \in M(d_i)\}$ – that is, the data types in d_i's data model. Sanitizers for each d_j must then be identified, and the same process that was used for d_i is recursively applied to any d_j for which a sanitizer is not available.

If Σ denotes the set of all trusted sanitizers available, the set $\sigma(\tau(d_i))$ of sanitizers needed to downgrade data d_i is

$$\sigma(\tau(d_i)) \quad = \quad \begin{cases} \{S(\tau(d_i))\} & : \quad S(\tau(d_i)) \in \Sigma \\ \bigcup\limits_{t \in T(d_i)} \sigma(t) & : \quad S(\tau(d_i)) \notin \Sigma \end{cases}$$

Data objects d_j for which no sanitizer is available can be marked as such, leaving them open to being blocked by a guard.

It should be noted that $\sigma(\tau(d_i))$ can be efficiently computed using a bottom-up parser [2], assuming the grammar corresponding to d_i's data model is context free.

C. Ordering constraints

The order in which different types of data are sanitized can substantially affect the result produced. For example, consider

a downgrading rule that redacts any data object that does not have a valid digital signature in the metadata, and another rule that removes the high-frequency components of digital images to blur them. If the second rule is performed before the first, the digital signature will be invalid, causing the entire image to be removed from the data prior to release. If the rules were performed in the other order, a suitably blurred image would be transmitted.

A monolithic downgrader collects all such constraints, determines an ordering that will satisfy all of them (if one exists), and then invokes the various sanitization functions as needed. A concern that arises in the case that the sanitization functionality is decomposed is whether this introduces an opportunity to violate the security policy by effecting the sanitization in a different order than required. In principle, this could be addressed by requiring each sanitizer to digitally sign its output. The sequence in which the data was sanitized can then be verified by a guard prior to releasing the data to a lower security level.

In practice, modern separation kernels include support for ensuring the sequencing of execution across multiple partitions [27]. A front end for the downgrading process can perform the constraint-solving needed to determine the required order of sanitization, as shown in Figure 6. In the first step, the set of ordering constraints $C(d_i)$ for sanitizing data d_i is calculated

$$C(d_i) = \{S(t_1) < S(t_2), S(t_3) < S(t_4), S(t_5) < S(t_6), \ldots\}$$

where $t_j \in T(d_i)$ and $S(t_1) < S(t_2)$ indicates that data of type t_1 must be sanitized before data of type t_2.

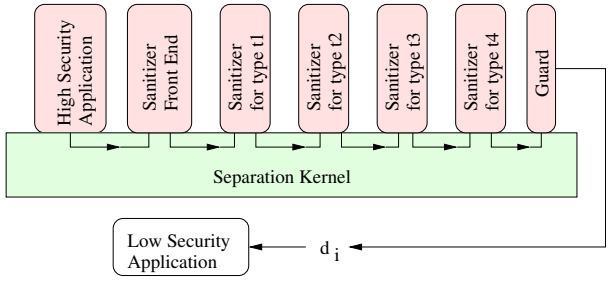

Fig. 6. A *trusted cascade* allows data to be sanitized in a predefined order that can be verified using support from the separation kernel. In this example, the goal is to sanitize data types in the order t_1 before t_2, t_2 before t_3, and t_3 before t_4.

If soft constraints are used, then the ordered set $O(d_i)$ of sanitizers $S(t_j) \in \sigma(\tau(d_i))$ that best fits the requirements is $O(d_i) = \{S(t_1), S(t_2), S(t_3) \ldots\}$ where $t_1, t_2, t_3, \ldots$ is the order in which types are to be sanitized. If hard constraints are used, it may not be possible to find a suitable ordered set of sanitizers.

$O(d_i)$ can effectively be communicated to the guard that is actually responsible for releasing the data from a higher to a low security level. The data is sent through a *trusted cascade* of sanitizers in the order specified in $O(d_i)$ by the front end. When the data emerges from the last sanitizer, the guard is able to rely on the separation kernel's sequencing to safely release the data.

D. Symbiotic opportunity

Systems that provide language-based information flow security are defined in abstract frameworks. Concrete realizations of them rely on substantial computational infrastructures such as host operating systems, virtual machines, security-typed runtimes, and compilers, as illustrated in Figure 7. Vulnerabilities in any of these are external to the threat model addressed by these systems but remain present in real deployments. We argue that the development of language-based security within the context of a MILS framework will enable them to address these concerns. In particular, this will entail proofs of composition of the security guarantees of the underlying infrastructure and the security-typed runtime.

Simultaneously, the availability of distributed language-based security within the MILS framework will enable the utilization of untrusted data sanitizers written in the supported language. The security-type system will then be able to implicitly provide the witnesses described in Section III-C. In particular, the task of ensuring that sensitive data does not leak can be localized to the declassification policy of the security-typed runtime rather than needing formal verification of the sanitizer itself.

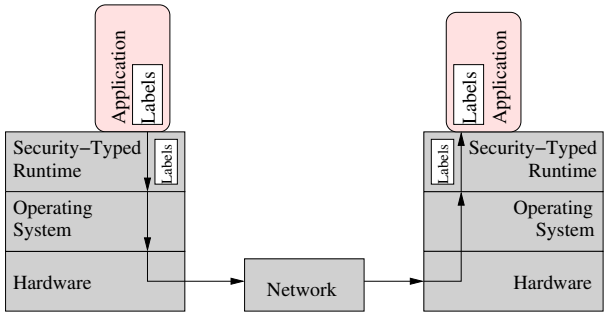

Fig. 7. In this example, the *decentralized label model* relies on the security-typed runtimes of two distributed hosts to enforce the same type system. Without this property, the policy defined by a principal at one host will not hold at the other host. Thus, the language-based information flow protection relies on the security of the underlying runtime, operating system, and hardware.

Finally, downgrading is being utilized for increasingly diverse data. If the same sanitization is applied, regardless of context, the most conservative fidelity reduction options must be chosen since the adversary is not known *a priori*. A policy-based approach [9] would allow the extent of sanitization to be selected dynamically, accounting for the *purposes* for which the data will be used, as well as the *anti-purposes* – that is, the uses that should specifically be prevented. The development of such downgrading functionality in the MILS framework will enable applications to extract maximum utility from the data without violating the system's security policy.

V. Conclusion

As the content and context of the data being transmitted in high assurance systems continues to increase in complexity, the cost and time to certify cross-domain solutions is growing rapidly. We argue that downgrading functionality should be

decomposed to the point where certification is economically viable, and that the resulting data sanitization primitives can be orchestrated to provide equivalent functionality to that provided by monolithic downgraders. Finally, we observe that the development of security-typed runtimes within the MILS framework will offer an opportunity to create untrusted data sanitizers that cannot leak information.

REFERENCES

[1] Nabil Adam and John Worthmann, Security-control methods for statistical databases: A comparative study, ACM Computing Surveys, Vol. 21(4), 1989.

[2] Alfred Aho, Ravi Sethi, and Jeffrey Ullman, Compilers: Principles, tools and techniques, Addison-Wesley, 1986.

[3] Jim Alves-Foss, W. Scott Harrison, Paul Oman, and Carol Taylor, The MILS architecture for high assurance embedded systems, International Journal of Embedded Systems, Vol. 2(3/4), 2006.

[4] Paul Beynon-Davies, Database systems, Macmillan, 2000.

[5] Carolyn Boettcher, Rance DeLong, John Rushby, and Wilmar Sifre, The MILS component integration approach to secure information sharing, 27th IEEE/AIAA Digital Avionics Systems Conference, 2008.

[6] Arthur Cebrowski and John Gartska, Net-centric warfare: Its origin and future, U.S. Naval Institute, 1998.

[7] Bee-Chung Chen, Daniel Kifer, Kristen LeFevre, and Ashwin Machanavajjhala, Privacy-preserving data publishing, Foundations and Trends in Databases, Vol. 2(1-2), 2009.

[8] Stephen Chong and Andrew C. Myers, Security policies for downgrading, 11th ACM Conference on Computer and Communications Security, 2004.

[9] Grit Denker, Ashish Gehani, Minyoung Kim, and David Hanz, Policy-based data downgrading: Toward a semantic framework and automated tools to balance need-to-protect and need-to-share policies, 11th IEEE International Symposium on Policies for Distributed Systems and Networks, 2010.

[10] Dorothy Denning and Peter Denning, Certification of programs for secure information flow, Communications of the ACM, Vol. 20(7), 1977.

[11] Josiah Dodds, A development environment and static analyses for GUARDOL - a language for the specification of high assurance guards, Master's Thesis, Kansas State University, 2010.

[12] Cynthia Dwork, Differential privacy: A survey of results, Theory and Applications of Models of Computation, Lecture Notes in Computer Science, Vol. 4978, Springer-Verlag, 2008.

[13] Boyd Fletcher, XML Data flow configuration file format specification, http://iase.disa.mil/cds/helpful_tools/dfcf-specification_1_2_11.pdf, 2008.

[14] Judith Froscher and Catherine Meadows, Achieving a trusted database management system using parallelism, Database Security II: Status and Prospects, North-Holland, 1989.

[15] Benjamin Fung, Ke Wang, Rui Chen, and Philip Yu, Privacy-preserving data publishing: A survey on recent developments, ACM Computing Surveys, 2010.

[16] GIG IA Architecture v1.1, https://www.us.army.mil/suite/kc/13000401

[17] Joseph Goguen and Jose Meseguer, Security policies and security models, 3rd IEEE Symposium on Security and Privacy, 1982.

[18] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer, A secure environment for untrusted helper applications, 6th USENIX Security Symposium, 1996.

[19] James Gosling, Bill Joy, and Guy Steele, Java language specification, Addison-Wesley, 1996.

[20] Boniface Hicks, David King, and Patrick McDaniel, Declassification with cryptographic functions in a security-typed language, Technical Report NAS-TR-0004-2005, Network and Security Center, Department of Computer Science, Pennsylvania State University, 2005.

[21] Michael Hicks, Stephen Tse, Boniface Hicks, and Steve Zdancewic, Dynamic updating of information-flow policies, International Workshop on Foundations of Computer Security, 2005.

[22] High Speed Guard, Raytheon, http://www.raytheon.com/capabilities/products/cybersecurity/highspeedguard/

[23] Myong Kang, Ira Moskowitz, and Stanley Chincheck, The pump: A decade of covert fun, 21st Annual Computer Security Applications Conference, 2005.

[24] Sameer Kumar and Kevin Moore, The evolution of Global Positioning System technology, Journal of Science Education and Technology, Vol. 11(1), 2002.

[25] Adam Lee, Parisa Tabriz, and Nikita Borisov, A privacy-preserving interdomain audit framework, 5th ACM Workshop on Privacy in Electronic Society, 2006.

[26] Ninghui Li, John Mitchell, and William Winsborough, Design of a role-based trust-management framework, IEEE Symposium on Security and Privacy, 2002.

[27] LynxSecure Separation Kernel-Hypervisor Version 4.0 User Guide, LynuxWorks, 2011.

[28] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam, l-diversity: Privacy beyond k-anonymity, ACM Transactions on Knowledge Discovery from Data, Vol. 1(1), 2007.

[29] Ashwin Machanavajjhala, Johannes Gehrke, and Michaela Goetz, Data publishing against realistic adversaries, Very Large Databases, Vol. 2(1), 2009.

[30] Andrew Myers and Barbara Liskov, Protecting privacy using the decentralized label model, ACM Transactions on Software Engineering and Methodology, Vol. 9(4), 2000.

[31] Nancy Reed, Dave Bryson, James Garriss, Steve Gosnell, Brook Heaton, Gary Huber, David Jacobs, Mary Pulvermacher, Salim Semy, Chad Smith, and John Standard, Security guards for the future Web, MITRE Technical Report MTR 04W0000092, 2004.

[32] John Rushby, Design and verification of secure systems, ACM Symposium on Operating System Principles, Vol. 15, 1981.

[33] Andrei Sabelfeld and Andrew Myers, Language-based information-flow security, IEEE Journal on Selected Areas in Communications, Vol. 21(1), 2003.

[34] Andrei Sabelfeld and Andrew Myers, A model for delimited information release, International Symposium on Software Security, Lecture Notes in Computer Science, Vol. 3233, Springer-Verlag, 2004.

[35] Andrei Sabelfeld and David Sands, Declassification: Dimensions and principles, Journal of Computer Security, Vol. 17(5), 2009.

[36] Richard Smith, Multilevel security, Handbook of Information Security, Wiley, 2006.

[37] Brian Snow, We need assurance!, 21st Annual Computer Security Applications Conference, 2005.

[38] Malcolm Stevens and Michael Pope, Data diodes, Technical Report ADC055868, Electronics and Surveillance Research Laboratory, Information Technology Division, Defense Science and Technology Organization, Canberra, Australia, 1995.

[39] Latanya Sweeney, k-anonymity: A model for protecting privacy, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 10(5), 2002.

[40] Trusted Services Engine, Galois, http://www.galois.com/files/TSE_Datasheet.pdf

[41] Tactical Cross Domain Solution, General Dynamics, http://www.gdc4s.com/documents/GD-TCDS-w.pdf

[42] Stephen Tse and Steve Zdancewic, A design for a security-typed language with certificate-based declassification, 14th European Symposium on Programming, Lecture Notes in Computer Science, Vol. 3444, Springer-Verlag, 2005.

[43] U.S. Fleet Forces Command Briefing, Training - Cross Domain Information Sharing (T-CDIS) Summit, U.S. Joint Forces Command, Suffolk, VA, 28 October 2009.

[44] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine, A sound type system for secure flow analysis, Journal of Computer Security, Vol. 4(3), 1996.

[45] Dennis Volpano, Secure introduction of one-way functions, 13th IEEE Computer Security Foundations Workshop, 2000.

[46] Daniel Wallach, Andew Appel, and Edward Felten, The security architecture formerly known as stack inspection: A security mechanism for language-based systems, ACM Transactions on Software Engineering and Methodology, Vol. 9(4), 2000.

[47] Jun Xu, Jinliang Fan, Mostafa Ammar, and Sue Moon, On the design and performance of prefix-preserving IP traffic trace anonymization, 1st ACM SIGCOM Workshop on Internet Measurement, 2001.

[48] Zhiqiang Yang, Sheng Zhong, and Rebecca Wright, Anonymity-preserving data collection, 11th ACM International Conference on Knowledge Discovery in Data Mining, 2005.