"Guarding the Flow: Tackling Main Issues with Information Flow Security and Mitigating Them with Advanced Approaches”

In the digital age, web applications have undoubtedly revolutionised our interconnected world by enabling a wide range of essential services, from online banking to healthcare solutions. However, this technological marvel comes at a price - the ever-looming threat of hacker attacks. [1]

The innate design of web applications introduces inherent vulnerabilities. Such event-driven applications are typically modelled as reactive programs [21], where a program is a set of event handlers, triggered by corresponding user input events. Shared storage allows the same data to be accessed by different event handlers (e.g., cookies) and organises the event handlers themselves (e.g., the DOM). These applications often include code from heterogeneous and untrusted sources and could potentially leak the users’ sensitive data to an adversary. To prevent such leaks, many runtime mechanisms have been developed for enforcing information flow control (IFC) policies [22], [23], [24], [25], [26], [27], most of which guarantee (variants of) noninterference, i.e., private data should not influence data sent on channels that are publicly observable [28]. In the realm of web applications, where the flow of information is crucial, policies must strike a balance: they must be finely detailed to address both unwanted and desired information flows. Drawing upon the established framework for controlled information release, as outlined by Sabelfeld and Sands [30], these policies should offer precise specifications. This involves determining what information can be disclosed (like the contents of a document), by specific individuals (such as owners or designated "friends"), along with conditions related to the "where" and "when" (such as making content public only after an owner marks it as such). Additionally, it's important to express these policies as locally as possible, ideally encapsulating them within each node in a network. These local policies can then be combined to form broader policies for more extensive systems, such as distributed networks of web applications. [29] It is unsurprising that six out of the ten most critical vulnerabilities in web applications relate to information-flow violations. These violations can compromise either integrity, where untrusted inputs infiltrate security-sensitive operations, or confidentiality, wherein private information is exposed to unauthorised observers. [13]

Over the past decade, extensive research has been dedicated to devising methods and algorithms for automatically detecting these information-flow violations in web applications. However, many of the approaches presented in research literature prove impractical for real-world industrial applications. Approaches based on complex type systems tend to lack broad adoption due to their intricacy and conservatism [2,3,4]. Similarly, methods founded on program slicing often fall short in terms of soundness [5] or scalability [6, 7]. [12]

A revealing study conducted by the Application Defense Center delved into the heart of this vulnerability conundrum. It scrutinised over 250 web applications from various domains, such as e-commerce, online banking, enterprise collaboration, and supply chain management. The chilling revelation was that a staggering 92% of these web applications were found vulnerable to some form of attack. It was a stark wake-up call, underscoring the pressing need for robust security measures. Despite common use of defences such as firewalls and

intrusion detection or prevention systems, hackers can access valuable proprietary and customer data, shut down websites and servers, defraud businesses, and introduce serious legal liability without being stopped or, in many cases ,even detected. [1]
In conclusion, the complex landscape of web application vulnerabilities and the evolving techniques of cyber attackers necessitate a multi-pronged approach. It is crucial to not only address the vulnerabilities that make these applications susceptible to attacks but also to design and implement robust security measures that can effectively mitigate the ever-present risks. As our dependence on web applications continues to grow, so does the urgency to fortify their defences and uphold the integrity of our digital infrastructure.
Safeguarding sensitive information within computing environments has been widely acknowledged as a complex and formidable challenge. All modern OSs include some form of access control to protect files from being read or modified by unauthorised users. Nevertheless, access controls prove inadequate in governing the dissemination of information once it has been unleashed for processing by a program. Similarly, cryptography provides strong confidentiality guarantees in open, possibly hostile environments like the Internet, but it is prohibitively expensive to perform nontrivial computations with encrypted data. Neither access control nor encryption provide complete solutions for protecting confidentiality.
A complementary approach, proposed more than thirty years ago, is to track and regulate the information flows of the system to prevent secret data from leaking to unauthorised parties. . This can be achieved through dynamic means, such as assigning security-level labels to data and subsequently extending these labels to all data derivatives, or through static methods, wherein the software handling the data is scrutinised to establish its adherence to predetermined data-related guidelines [16][17]. Static analysis is more precise analysis compared to dynamic as the analyst can see the entire program structure and all possible execution paths. However, static analysis can be computationally expensive, and it is not always possible to find all security vulnerabilities. Dynamic analysis on the other hand however allows for a more comprehensive analysis, as the analyst can see how the program actually behaves. However, dynamic analysis can be less precise than static analysis, as it is only possible to see the execution paths that are actually taken. Arguably, a mostly static approach (perhaps augmented with some dynamic checks) is the most promising way of enforcing information-flow policies. [14]

Two main approaches used by the authors were Language based approaches [10] which focuses on a specific programming language. Since they work on the level of language syntax, they often achieve an impressive degree of automation. For example, Jif [3] extends Java with security labels for data values, and enforces security via a combination of static and run-time checking. It supports control over who may declassify information, but not what is declassified. Joana [33] checks noninterference of Java programs via static program analysis. Control of declassification is limited to where in the program declassification may occur. Jeeves [34] extends a core language for functionality with a language for flexible security policies. Secure multi-execution [35] is a run-time enforcement technique where multiple copies of a program are executed, one for each security level, controlling the information flows between the levels. Secure program partitioning [36,37] produces a distribution of the code and data in a program onto different hosts according to their different trust levels (e.g.

trusted web server and untrusted client-side browser). While each of the aforementioned approaches supports certain forms of declassification, they do not consider the setting and the kind of compositional reasoning that we aim for: given a network of communicating, nondeterministic systems, derive a global, complex information flow security property from the local security properties of the components. DStar [38] and Fabric [39] do consider distributed systems, but only support control over which processes or principles may declassify information, not what is declassified. The WHAT&WHERE security property [40] does allow control over what is declassified by concurrent programs, but only in a non-interactive setting, not suitable for web applications.
By contrast, system-based approaches work with security properties expressed directly on the semantics, on variants of event systems or input/output-automata. The system-based approach to IFS uses a separate system to enforce security policies. This system is typically implemented as a runtime monitor that observes the execution of the program and checks for violations of security policies. The system-based approach can be used to enforce a wider range of security policies than the language-based approach, but it is also more complex and less precise. Early work following this approach [41] has observed that even seemingly strong security properties are not preserved under composition in general. Consequently, comprehensive frameworks have been developed for the composition of security properties in various settings, e.g. event systems [42], reactive systems [43] or process calculi [44]. However, these frameworks do not consider declassification, except for very specific notions such as intransitive noninterference [45]. Chong and van der Meyden [46] discuss information flow policies (called architectures), where filter functions are used to restrict what information may flow between domains, together with an interpretation of the resulting security properties in terms of an epistemic logic. However, they do not consider compositional reasoning in our sense, i.e. composing the security properties of multiple systems. The same applies to the work on temporal logics and model checking approaches for hyperproperties [47], of which information flow security properties are an instance. Greiner and Grahl [48] present a compositionality result that supports what-declassification control, specified in terms of equivalence relations on communication events, but it cannot express dynamically changing confidentiality requirements — as needed for web applications in general and for CoSMeDis in particular: for example, whether a given post p by user u is confidential for an observer depends on the visibility setting of p and/or the friendship status between u and the observer.

While there are other approaches like the User-Centric approach in which focus is on giving users more control over the information they share and receive. This can involve providing users with clear privacy settings, allowing them to customise what information is shared with different entities, and providing notifications about data access. [31], Decentralised Information Flow Control (DIFC) approach allows application writers to control how data flows between the pieces of an application and the outside world. When applied to privacy, DIFC allows untrusted applications to compute with private data while trusted security code controls the release of that data while when applied to integrity, DIFC allows trusted code to protect untrusted applications from unexpected malicious inputs. They treat the server as black box and track the data as the response to a request is being constructed. The security architecture is made of a security gateway and an operating system library which tags data as it is being used by the web application. The core concept is to centralise all security decisions in the gateway and prevent unwanted data access. [32]

In a recent and all-encompassing study conducted by Sabelfeld and Myers [10], an extensive collection of 147 references to publications concerning information-flow security was compiled. The majority of these works delve into the intricate exploration and refinement of various interpretations of noninterference, a foundational property in information-flow that essentially demands the insulation of secret data from influencing the publicly observable behaviour of a system. According to Sabelfeld and Myers, typed systems are best for implementing static analysis and language semantics. In a security-typed language, the types of program variables and expressions are augmented with annotations that specify policies on the use of the typed data. These security policies are then enforced by compile-time type checking, and, thus, add little or no run-time overhead. Like ordinary type checking, security-type checking is also inherently compositional: secure subsystems combine to form a larger secure system as long as the external type signatures of the subsystems agree. Many of the remaining studies expound upon methodologies for implementing information-flow regulations through the application of program analysis techniques. Despite this substantial repository of scholarly work and the continual attention it garners from the research community, it remains notable that mechanisms for enforcing information-flow through these approaches have not achieved widespread, or even limited, adoption. Li and Zdancewic [11] present a similar language based approach to enforcing confidentiality and integrity of data. They propose a security-typed PHP-like scripting language to address information-flow control in web applications by elevating queries to first class status in the language and enforce information flow policies statically. They intend to improve web application security by ensuring that the application conforms to a set of information flow assertions. In contrast to Li-Zdancewic, Yip et. al. [15] works with existing languages and enforces policies dynamically in the language runtime.

Another study which uses dynamic analysis as its basis was done by Staicu et. al. [8] to address the questions on how common different kinds of flows are in real-world programs, how important these flows are to enforce security policies, and how costly it is to consider these flows. These questions are addressed by using 56 real world Javascript programs that suffer from various security problems, such as code injection vulnerabilities, memory leaks, and privacy leaks. So at a language level, a program may propogate information via two kinds of information flows: Explicit Flows (happens whenever sensitive information is passed by an assignment statement or into a sink) [18] and Implicit Flows (arise via control-flow structures of program, when the flow of control depends on the sensitive value) [18]. For a dynamic information flow analysis, implicit flows can be further classified into flows that happen because a particular branch is executed, so-called observable implicit flows [19], and flows that happen because a particular branch is not executed, so called hidden implicit flows [19]. Ideally, an information flow analysis should consider all three kinds of flows. In fact, there exists a large body of work on static, dynamic, hybrid (both static and dynamic analysis), and multi-execution techniques to prevent explicit and implicit flows.

However, so far these tools have seen little use in practice, despite the strong security guarantees that they provide. In contrast, a lightweight form of information flow analysis called taint analysis is widely used in computer security [20]. Taint analysis is a pure data dependency analysis that only tracks explicit flows, ignoring any control flow dependencies. So from the study done by Staicu et. al. it was found that implicit flows are expensive to track in terms of permissiveness, label creep, and runtime overhead. This study underscores the importance of comprehensive dynamic information flow analysis to ensure security and privacy in real-life applications. The findings reveal that considering explicit and observable implicit flows is crucial for detecting vulnerabilities effectively, but tracking hidden implicit flows requires careful consideration due to their impact on analysis efficiency. Proper management of sensitive data and monitoring strategies is essential to strike a balance between accuracy and performance during program execution. It finds that taint tracking is sufficient for most of the 56 studied problems, while for some observable tracking is needed.

Model checking is a technique that can be used to check for security vulnerabilities in a program by exhaustively exploring all possible execution paths. This is a very powerful approach, but it is also very computationally expensive. Livshits et. al. [9] introduces PQL (Program Query Language), a concise and declarative language designed to define information flow patterns. Livshits et. al. have developed a static context-sensitive, but flow-insensitive information flow tracking analysis similar to [17] that can be used to find all the vulnerabilities in the program. Livshits et. al. uses a Model checking system to analyse more precisely in the event that analysis generates too many warnings. Model checking is also used to automatically generate the input vector that exposes the vulnerability. Any remaining behaviour these static analyses have not isolated may be checked dynamically. The result of the static analysis is used to optimise these dynamic checks. The experimental results from the study show that the language used in the research is powerful and versatile enough to effectively describe a significant number of security vulnerabilities in a concise manner. The researchers conducted analyses on more than nine applications and identified a total of 30 serious security vulnerabilities. Furthermore, the study's dynamic checker, which is a tool or mechanism used to perform dynamic information flow analysis during program execution, demonstrated its capability to not only detect security attacks as they happened but also automatically recover from these attacks. This means that the dynamic checker was able to promptly identify and respond to security breaches or attempted exploits, mitigating their impact and restoring the system to a secure state. It is shown that it can automatically find a large number of security vulnerabilities in real-life web applications through the synergism of a new language for describing information flow, context-sensitive pointer alias analysis, dynamic monitoring, and model checking.

Another research done on taint analysis by Tripp et. al. [12] addresses the challenges of false findings and scalability that traditional methods like program slicing and type system encounters. Tripp et. al [12] developed an analysis tool, named ANDROMEDA, which uses this approach by enabling on-demand data-flow computations that are efficient, accurate, and adaptable through incremental analysis. ANDROMEDA's innovative approach centres on tracking vulnerable information flows in a demand-driven manner, eliminating the need for a comprehensive representation of the entire application. It constructs a call-graph representation through intraprocedural type inference. When aliasing relationships need to be established due to data flow into the heap, ANDROMEDA employs specific aliasing queries focused on the particular flow, avoiding the necessity for full-program pointer analysis. This strategy ensures effective and efficient scanning for large applications, concentrating modelling efforts where necessary and facilitating incremental analysis even after code changes. Viewed differently, ANDROMEDA functions as an extended type system, where an automated, context-sensitive, interprocedural, and incremental inference engine automatically assigns security annotations to program points and propagates them.

The papers provide a good overview of the state of the art in information flow security of web applications. They discuss a number of different techniques for enforcing information flow security, and they prevent a number of different malicious attack scenarios or leakage. However, there is still more work to be done in this area.
In the intricate realm of web application security, where the flow of information is both essential and precarious, a multi-faceted approach emerges as paramount. From language-based methodologies that automate information flow control to system-based strategies that directly address semantics, each approach offers unique strengths. Dynamic analyses, like taint analysis and dynamic checkers, contribute to real-time vulnerability detection, while user-centric and decentralised approaches empower users and regulate data flows within applications. Challenges persist in terms of adoption and scalability, as exemplified by model checking and innovative tools like ANDROMEDA. Through comprehensive research and experimentation, the complexities of web application vulnerabilities become evident. As the digital landscape evolves, safeguarding sensitive data remains a pressing concern. The journey toward fortified web application defences demands a harmonious integration of these diverse approaches, collaborative innovation, and unwavering dedication to upholding the integrity of our digital infrastructure.

References

[1] *WebCohort, Inc. Only 10% of Web applications are secured against common hacking techniques. http://www.imperva.com/ company/news/2004-feb-02.html, 2004.*

[2] *Volpano, D., Irvine, C., Smith, G.: A Sound Type System for Secure Flow Analysis. JCS 4(2-3) (1996)*

[3] *Myers, A.C.: JFlow: Practical Mostly-static Information Flow Control. In: POPL (1999)*

[4] *Shankar, U., Talwar, K., Foster, J.S., Wagner, D.: Detecting Format String Vulnerabilities with Type Qualifiers. In: USENIX Security (2001)*

[5] *Tripp, O., Pistoia, M., Fink, S.J., Sridharan, M., Weisman, O.: TAJ: Effective Taint Analysis of Web Applications. In: PLDI (2009)*

[6] *Hammer, C., Krinke, J., Snelting, G.: Information Flow Control for Java Based on Path Conditions in Dependence Graphs. In: S&P (2006)*

[7] *Snelting, G., Robschink, T., Krinke, J.: Efficent Path Conditions in Dependence Graphs for Software Safety Analysis. TOSEM, 15(4) (2006)*

[8] *Cristian-Alexandru Staicu, Daniel Schoepe, Musard Balliu, Michael Pradel, and Andrei Sabelfeld. 2019. An Empirical Study of Information Flows in Real-World JavaScript. In Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security (PLAS'19). Association for Computing Machinery, New York, NY, USA, 45–59. https://doi.org/10.1145/3338504.3357339*

[9] *Lam, M. S., Martin, M., Livshits, B., & Whaley, J. (2008). Securing web applications with static and dynamic information flow tracking. In Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (pp. 3–12). https://doi.org/10.1145/1328408.1328410*

[10] *A. Sabelfeld and A. C. Myers, "Language-based information-flow security," in IEEE Journal on Selected Areas in Communications, vol. 21, no. 1, pp. 5-19, Jan. 2003, doi: 10.1109/JSAC.2002.806121.*

[11] *Peng Li and S. Zdancewic, "Practical information flow control in Web-based information systems," 18th IEEE Computer Security Foundations Workshop (CSFW'05), Aix-en-Provence, France, 2005, pp. 2-15, doi: 10.1109/CSFW.2005.23.*

[12] *Tripp, Omer & Pistoia, Marco & Cousot, Patrick & Cousot, Radhia & Guarnieri, Salvatore. (2013). ANDROMEDA: accurate and scalable security analysis of web applications. 210-225. 10.1007/978-3-642-37057-1_15.*

[13] *M. McCall, A. Bichhawat and L. Jia, "Compositional Information Flow Monitoring for Reactive Programs," 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P), Genoa, Italy, 2022, pp. 467-486, doi: 10.1109/EuroSP53844.2022.00036.*

[14] *Zdancewic, Steve. “Challenges for Information-flow Security.” (2004).*

[15] *Alexander Yip, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. Improving application security with data flow assertions. In Proceedings of the 193 22nd acm Symposium on Operating Systems Principles (sosp’09), pages 291–304, Big Sky, mt, usa, October 2009. Acm.*

[16] *K. Chen, O. Arias, Q. Deng, D. Oliveira, X. Guo and Y. Jin, "FineDIFT: Fine-Grained Dynamic Information Flow Tracking for Data-Flow Integrity Using Coprocessor," in IEEE*

Transactions on Information Forensics and Security, vol. 17, pp. 559-573, 2022, doi: 10.1109/TIFS.2022.3144868.

[17] *Liu, Yin and Ana L. Milanova. "Static Information Flow Analysis for Java." (2008).*

[18] *Dorothy E. Denning and Peter J. Denning. 1977. Certification of programs for secure information flow. Commun. ACM 20, 7 (1977), 504–513*

[19] *Musard Balliu, Daniel Schoepe, and Andrei Sabelfeld. 2017. We Are Family: Relating Information-Flow Trackers. In Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I. 124–145.*

[20] *Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In IEEE S&P.*

[21] A. Bohannon, B. C. Pierce, V. Sjoberg, S. Weirich, and ¨ S. Zdancewic. Reactive noninterference. In ACM CCS, 2009.

[22] I. Bastys, M. Balliu, and A. Sabelfeld. If this then what? controlling flows in IoT apps. In ACM CCS, 2018.

[23] I. Bastys, F. Piessens, and A. Sabelfeld. Tracking information flow via delayed output. In NordSec, 2018.

[24] L. Bauer, S. Cai, L. Jia, T. Passaro, M. Stroucken, and Y. Tian. Run-time monitoring and formal analysis of information flows in Chromium. In NDSS, 2015.

[25] A. Bichhawat, V. Rajani, J. Jain, D. Garg, and C. Hammer. WebPol: Fine-grained information flow policies for web browsers. In ESORICS, 2017

[26] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens. FlowFox: a web browser with flexible and precise information flow control. In ACM CCS, 2012

[27] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In USENIX OSDI, 2010

[28] *J. A. Goguen and J. Meseguer. Security policies and security models. In IEEE Symposium on Security and Privacy, pages 11– 20, 1982.*

[29] Bauereiß, Thomas, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. "Compositional Verification of Information Flow Security for Distributed Web Applications."

[30] A. Sabelfeld and D. Sands, "Declassification: Dimensions and principles," Journal of Computer Security, vol. 17, no. 5, pp. 517–548, 2009.

[31] C.G. Sorensen, L. Pesonen, S. Fountas, P. Suomi, D. Bochtis, P. Bildsøe, S.M. Pedersen, A user-centric approach for information modelling in arable farming, Computers and Electronics in Agriculture, Volume 73, Issue 1, 2010, Pages 44-55, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2010.04.003.

[32] Jean-Jacques Dubray "Securing the Web with Decentralised Information Flow Control" https://www.infoq.com/news/2008/08/securing-the-web/

[33] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," Int. J. Inf. Sec., vol. 8, no. 6, pp. 399–422, 2009.

[34] J. Yang, K. Yessenov, and A. Solar-Lezama, "A language for automatically enforcing privacy policies," in POPL, 2012, pp. 85–96.

[35] D. Devriese and F. Piessens, "Noninterference through secure multiexecution," in IEEE Security and Privacy, 2010, pp. 109–124.

[36] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng, "Building secure web applications with automatic partitioning," Commun. ACM, vol. 52, no. 2, pp. 79–87, 2009.

[37] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers, "Secure program partitioning," ACM Trans. Comput. Syst., vol. 20, no. 3, pp. 283–328, 2002.

[38] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières, "Securing distributed systems with information flow control," in NSDI, 2008, pp. 293–308.

[39] J. Liu, M. D. George, K. Vikram, X. Qi, L. Waye, and A. C. Myers, "Fabric: a platform for secure distributed computation and storage," in SOSP, 2009, pp. 321–334.

[40] A. Lux, H. Mantel, and M. Perner, "Scheduler-independent declassification," in MPC, 2012, pp. 25–47.

[41] D. McCullough, "Specifications for multi-level security and a hook-up property," in IEEE Security and Privacy, 1987.

[42] ——, "On the composition of secure systems," in IEEE Security and Privacy, 2002, pp. 88–101.

[43] W. Rafnsson and A. Sabelfeld, "Compositional information-flow security for interactive systems," in CSF, 2014, pp. 277–292.

[44] A. Bossi, D. Macedonio, C. Piazza, and S. Rossi, "Information flow in secure contexts," Journal of Computer Security, vol. 13, no. 3, pp. 391–422, 2005.

[45]] H. Mantel, "Information flow control and applications - bridging a gap," in FME, 2001, pp. 153–172.

[46] S. Chong and R. V. D. Meyden, "Using Architecture to Reason About Information Security," ACM Trans. Inf. Syst. Secur., vol. 18, no. 2, pp. 8:1–8:30, Dec. 2015.

[47] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in POST, 2014, pp. 265–284.

[48] S. Greiner and D. Grahl, "Non-interference with what-declassification in component-based systems," in CSF, 2016, to appear.