Randomized watermarks for structured programs

Lucila Bento*, Davidson Boccardo†, Raphael Machado†, Vinícius Pereira de Sá*, and Jayme Szwarcfiter*†
*Universidade Federal do Rio de Janeiro – UFRJ
†Instituto Nacional de Metrologia, Qualidade e Tecnologia – INMETRO
Email: lucilamsbento@gmail.com, {drboccardo, rcmachado}@inmetro.gov.br, vigusmao@dcc.ufrj.br, jayme@nce.ufrj.br

I. Introduction

Watermarking an object is the act of embedding an identifier of authorship/ownership within that object, so to discourage illegal copying. Some years ago, this idea was leveraged to the context of software protection. Different approaches to software watermarking have been devised to date. *Graph-based* watermarking schemes consist of (i) encoding/decoding algorithms (*codecs*) that translate the identification data (the *key*) onto (and back from) some special kind of graph; and (ii) embedding/extracting algorithms to insert/retrieve the watermark graph into/from the program.

Venkatesan et al. [5] proposed the first graph-based watermarking scheme in which the watermark graph is embedded into the control flow graph (CFG[1]) of the software by injecting dummy code into the original program. Recently, Chroni and Nikolopoulos proposed a codec [3] in which the key is encoded as an instance of the reducible permutation graphs introduced by Collberg et al. [4]. Such graphs have been proved to allow for the detection of (and recovery from) malicious attacks in the form of $k \leq 2$ edge removals [1].

Other than the standard desirable properties of a watermark (*small size*, *resilience* to attacks, and computational *efficiency*), ideally a watermarking scheme should be able to generate reasonably different graphs to encode the same key (*diversity*). Finally, an important feature in a watermark is the ability to be disguised into the code (*stealthiness*), preventing that it is easily found and removed after a reverse engineering endeavour undertook by a malicious party. Watermarks which can only make their way into the code with the aid of rather artificial instructions such as *goto* statements may give rise to suspicion, improving an attacker's odds of success.

II. Structured Watermarks

We propose a new codec for graph-based software watermarking with the following main properties:

- *Diversity*. The encoding algorithm employs randomization to produce distinct watermarks for the same key upon different executions. This feature makes it less likely that a watermark can be spotted by brute force comparisons among different watermarked programs of the same author.
- *Stealthiness*. The dummy code to be injected is *structured*—as defined by Dijkstra [2]—, which distinguishes it from all existing graph-based watermarks we know of (where *goto* statements are called for). Thus, when protecting structured software, the resulting watermarked CFG will belong to the class of Dijkstra graphs [2].
- *Small size*. The watermark has just $n + 2$ vertices and $2n + 1$ edges, where n is the bit-length of the key.
- *Resilience*. There is a one-to-one correspondence between the edges of the watermark and the bits of the encoded key, hence distortive attacks (whereby the watermark is damaged, but not removed) can be detected after the graph-to-key decoding process, and the correction of any flipped bits—up to some predefined number—can be carried out by standard error-correction techniques.
- *Efficiency*. Both encoding and decoding run in linear time.

A rough sketch of the algorithm's central idea follows. Initialize a watermark graph G as a directed path $P : 1, 2, \ldots, n+2$, then add an edge $v \to r(v)$ for each vertex $v \in \{2, \ldots, n\}$, where $r(v)$ is chosen uniformly at random among all *candidates* for $r(v)$. A vertex $w \in V(G) \setminus \{v\}$ is a candidate $r(v)$ if (i) the distance between v and w in P has the same parity as the vth bit in the binary key; and (ii) it does not violate structured program constraints. Decoding is straightforward, since the first bit in the key is always a "1" (leading zeroes are removed), and the parity of the difference between the endpoints of each edge in $G \setminus P$ indicates each remaining bit.

[1]The CFG represents all possible sequences of computation of the program's instructions in the form of a directed graph whose vertices are the blocks of strictly sequential code, and whose edges indicate possible precedence relations between those blocks.

References

[1] Lucila Bento, Davidson Boccardo, Raphael Machado, Vinícius Pereira de Sá, and Jayme Szwarcfiter. 2013. Towards a Provably Resilient Scheme for Graph-Based Watermarking. In *39th International Workshop Graph-Theoretic Concepts in Computer Science* (WG 2013). Springer, Lübeck, Germany, 50–63.

[2] Lucila Bento, Davidson Boccardo, Raphael Machado, Vinícius Pereira de Sá, and Jayme Szwarcfiter. 2015. The Graphs of Structured Programming. In *13th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*.

[3] Maria Chroni and Stavros Nikolopoulos. 2012. An Efficient Graph Codec System for Software Watermarking. In *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops* (COMPSACW'12). IEEE Computer Society, Washington, DC, USA, 595–600.

[4] Christian Collberg, Stephen Kobourov, Edward Carter, and Clark Thomborson. 2003. Error-Correcting graphs for software watermarking. In *Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science*, 156–167.

[5] Ramarathnam Venkatesan, Vijay V. Vazirani, and Saurabh Sinha. 2001. A Graph Theoretic Approach to Software Watermarking. In *Proceedings of the 4th International Workshop on Information Hiding* (IHW '01), Ira S. Moskowitz (Ed.). Springer-Verlag, London, UK, UK, 157–168.