Analyse du flot de données dans les codes binaires malveillants. Etude de la cartographie des fonctionnalités et leurs corrélations.

Nom de l'encadrant : Jean-Yves Marion

Résumé

L'équipe Carbone, dans le cadre du Laboratoire de Haute Sécurité (LHS), a développé une méthode originale d'analyse de codes binaires, dite par *analyse morphologique*. Cette méthode permet de détecter des similarités dans des codes. et de détecter des codes malveillants (virus, rançongiciels, malwares…). L'objectif de la thèse est de reconstruire le graphe de flot d'information d'un code binaire obfusqué pour reconstituer correctement la cartographie des fonctionnalités utilisées dans un code malveillant.

Le contexte de la détection des codes malveillants

Les compagnies d'Anti-Virus sont assez discrètes sur les méthodes de détection employées. Ceci dit, la technique classique de détection de code malveillant est d'attribuer à chacun une signature qui caractérise le malware en question. Une signature est une expression régulière, souvent réduite à une simple suite d'octets, qui identifie un malware. Classiquement on utilise une règle Yara [5] qui s'exprime ainsi (cas du rançongiciel Cerber) {68 3B DB 00 00 ?? ?? ?? ?? 00 ?? FF 15} où ?? correspond à un caractère quelconque. Tout fichier/binaire ou code exécuté en mémoire contenant cette signature, c'est à dire une suite d'octets satisfaisant l'expression régulière, est alors considéré infecté. Etant donnée une base de données de signatures, le moteur de détection est alors la partie de l'anti-virus chargée de rechercher une de ces signatures à l'intérieur des programmes. L'avantage de cette approche est sa rapidité et le faible taux de faux-positif. Les inconvénients tiennent essentiellement (i) dans le fait que cette technique n'est pas capable de détecter des variantes ou des mutations d'un code malveillant connu, et donc elle est à fortiori incapable d'identifier une nouvelle attaque, et (ii) dans le fait que la construction d'une signature est encore souvent faite manuellement.

Pour parer à ces difficultés, nous avons développé une approche dite par analyse morphologique. L'analyse morphologique s'appuie sur la reconstruction d'une abstraction du graphe de contrôle de flot (CFG). Cette reconstruction nécessite de combiner une analyse dynamique et une analyse statique. L'analyse dynamique consiste à exécuter dans un environnement sécurisé un programme en calquant l'exécution sur un modèle de vagues d'exécution pour tenir des obfuscations liées aux auto-modifications de code le plus souvent produites par des packers [2] ou encore sur le modèle proposé dans cet article [4]. L'analyse statique consiste à désassembler chaque vague de code et à reconstruire le graphe de contrôle de flot. La thèse en cours de Sylvain Cecchetto nous apporte de nouveaux outils d'analyse symbolique pour améliorer la correction du graphe de contrôle de flot obtenu. Ensuite, le graphe de flot est abstrait et découpé en petits graphes appelées sites. Les sites forment une base de comportement qui permet d'identifier un code malveillant.

Etat de l'art de la recherche de fonctionnalité

Au modèle de détection par base de signatures ou de comportement, on pourrait proposer une autre approche qui consisterait à identifier les fonctionnalités implantées dans un

code à analyser et à déterminer les relations entre ces différentes fonctionnalités. Dans un premier temps, cela apporterait une aide considérable à la rétro-conception. Pour mémoire, IDA, le state-of-the-art désassembleur, regarde juste le texte de l'en tête d'une fonction correctement compilée. Dans un second temps, cela permettrait de prédire un comportement malicieux. L'exemple serait une application qui contient un module de communication qui envoie des messages chiffrés avec un algorithme particulier comme Base64 (employé par APT28).
Sur du code x86 obfusqué, ce problème est dur. Il s'agit de retrouver la sémantique d'un code binaire obfusqué. Il y a peu d'approches. Citons en deux. La première consiste à considérer ce problème comme un problème d'interpolation [3]. Elle a été appliquée avec succès à l'identification des fonctions cryptographiques. Ceci étant, cette approche ne peut pas se généraliser à une autre classe de fonctions. La seconde approche [5] consiste à utiliser une analyse symbolique. Elle a aussi été appliquée pour identifier des fonctions de cryptographie. Ceci dit, la minutie de l'approche la rend, à ce jour, impraticable.

Sujet de thèse

L'analyse morphologique semble pouvoir apporter une solution. Si le découpage des sites n'est pas assez fin pour identifier les fonctionnalités implémentées, il s'avère qu'il est possible d'identifier un certain nombre de fonctions cryptographiques voire d'identifier le compilateur et les options utilisées. Dés lors, le plan de recherche est assez clair. Il s'agit de définir une méthode pour collecter un ensemble de sites qui vont caractériser une fonctionnalité, et pas seulement des fonctions cryptographiques comme dans les précédents travaux. Aujourd'hui, nous disposons des outils d'extraction du graphe de contrôle de flot. De ce fait, l'autre « variable » sur laquelle nous pouvons « jouer » est le flot d'information. Si cette approche a été partiellement suivie par [5], nous avons deux atouts : (i) nous savons construire des abstractions qui préservent la sémantique et (ii) nous devrions pouvoir appliquer la notion de site sur le (multi-)graphe de flot de données. Autrement dit, l'enjeu est de reconstruire le graphe de flot de données et de le découper de telle sorte à ce que, combiné avec les sites du graphe de flot de contrôle, nous soyons capables d'identifier une fonctionnalité. Une fois ce problème résolu, il ne restera pus qu'à déterminer les corrélations entre les différentes fonctionnalités identifiées. L'objectif est d'avoir une cartographie des fonctions implantées dans un programme et leurs corrélations. Une part non-négligeable du travail sera de valider expérimentalement l'approche.

Travaux cités

[1] Bardin, S., R. David, et J-Y Marion. «Backward-Bounded DSE: Targeting Infeasibility Questions on Obfuscated codes.» *IEEE Symposium on Security and Privacy (SP)*, 2017.
[2] Bonfante, G., J. Fernandez, J-Y Marion, B. Rouxel, F. Sabatier, et A. Thierry. «Codisasm: Medium scale concatic disassembly of self- modifying binaries with overlapping instructions.» *CCS*, 2015.
[3] Calvet, J., J. Fernandez, et J-Y Marion. «Aligot : cryptographic function identification in obfuscated binary programs.» *ACM Conference on Computer and Communications Security - CCS* , 2012.
[4] Cheng, B., et al. «Towards Paving the Way for Large-Scale Windows Malware Analysis: Generic Binary Unpacking with Orders-of-Magnitude Performance Boos.» *ACM Conference on Computer and Communications Security (CCS).*

[5] Lestringant, P. «Identification d'algorithmes Cryptographiques dans du code natif.» doctorat Université de Rennes 1.
[6] Yara. *http://virustotal.github.io/yara/.*