Voice over IP Vulnerability Assessment

Humberto Abdelnur

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Architecture de Sécurité sur la Voix sur IP

THÈSE

présentée et soutenue publiquement le 30 Mars 2009

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Humberto Jorge Abdelnur

Composition du jury

Président :	Dominique Méry
Rapporteurs:	Ana Cavalli Marc Dacier
Examinateurs:	Olivier Festor Radu State Stéphane Ubéda

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Dedicated to all those who had an influence on my education,
specially to my parents.

And to my family and friends.

Acknowledgments

I am not a man who can easily express his feelings, for which I guess this has been the hardest thing to write in my thesis (even if it does not look like).

I would like to give my most friendly gratefulness to my supervisors Radu and Olivier, for giving me the opportunity to come here and work with them. For the countless hours of discussions, the unlimited patience they had, their incredible motivation and support, their constant encouragement and specially for their sincere friendship.

I am thankful to Ana Cavalli, Marc Dacier, Dominique Méry and Stéphane Ubéda for honor me with their presence as part of my jury. Specially, I really appreciated all the critics, comments and improvements made by the jury in order to improve the quality of this manuscript.

A huge thanks to all my friends, those that I met here in France, those that moved along with me and those that still are in Argentina, for their beautiful friendship which always kept me encouraged to continue in spite of our distance.

I specially dedicate this thesis to my parents, I thank them for been so strict during my growing, for teaching me what is right and what is wrong, for all the effort they made for giving me every possible opportunity and, now, for relaxing and enjoy all the time that we can spend together. This won't be completed without thanking to my brother Gusti for the hours that he spent teaching me from biology to ... "toys bricolage", to my sister who gave me all the "galletas" that I ever asked her for and to my brother Alecito who has been my tutor over all my studies. And of course Checho, Anne and Silvi thank you for making me and my family happier and for giving us beautiful nieces and nephews Guada, Fosforito, Uggi, Gasparin and the one coming :) I am deeply grateful to the city of Nancy for the ilimited number of rainy days! Specially for metting who took my patience to the limits but made me every single day even happier. I love you Ale and Lauti.

Humberto

Contents

List of Figures

Une méthode d'évaluation de la sécurité des services voix sur IP

1 Introduction

Note

Ce chapitre est une synthèse en français des me travaux de thèse. Il résume les travaux présentés plus en détails dans les autres chapitres en anglais du manuscrit.

1.1 Contexte

La voix sur IP s'impose aujourd'hui comme l'un des services majeurs de l'Internet actuel et ses protocoles sont au cœur de l'évolution vers l'Internet du futur. Son déploiement en plein essor s'accompagne malheureusement d'une explosition du nombre de vulnérabilités liées, vulnérabilités pouvant mener à des attaques et à des détournements majeurs. Déjà soumis aux vulnérabilités inhénrentes à la couche IP sur laquelle ce service s'appuie, il porte en lui de nouvelles menaces pour la sécurité des systèmes informatiques comme nous l'avons démontré dans [12]. Afin d'améliorer la sécurité de ces services, il convient de fournir aux experts du domaine et aux développeurs des applications qui y sont déployées, des solutions capables d'automatiser le processus de découverte de vulnérabilités ainsi que les moyens pour vérifier que ces vulnérabilités sont bien couvertes par des protectoins efficaces dans des déploiements réels.

Nos travaux se placent dans cette perspective et portent sur la conception de tels environnements pour des systèmes communicants. Nous avons instancié nos approches sur le protocole SIP décrit dans la section 2.

1.2 Contributions

Nos contributions portent sur 3 éléments essentiels de l'analyse de sécurité de services voix sur IP. La première contribution est une architecture intégrée d'analyse de sécurité VoIP. Cette architecture permet d'automatiser le processus d'audit de sécurité d'une infrastructure voix sur IP et assure l'intégration et l'interopérabilité des composnats via un modèle d'information unifié. Elle est décrite dans la section 3. La seconde contribution de la thèse est un nouveau modèle de test de vulnérabilités capable d'aller tester par fuzzing un équipement dans des états avancés de son protocole d'exécution. Cette contribution est présentée dans la section 4. La troisième contribution porte sur une nouvelle méthode de fingerprinting d'équipements par analyse structurelle des messages. Cette méthode est décrite dans la section 5. Une synthèse des contributions est donnée dans la section 6.

1.3 Publications

Les travaux présentés dans la thèse ont fait l'objet des publications suivantes :

[12] H. Abdelnur, R. State, and O. Festor. Failles et VoIP. In MISC Magazine - Edition française: Multi-System & Internet Security Cookbook. Misc #39, Septembre/Octobre 2008.

[9] H. J. Abdelnur, R. State, I. Chrisment, and C. Popi. Assessing the security of VoIP Services. In The 10th IFIP/IEEE Symposium on Integrated Management (IM 2007), Munich, Germany, May 2007.

[8] H. Abdelnur, V. Cridlig, R. State, and O. Festor. VoIP Security Assessment: Methods and Tools. In 1st IEEE Workshop on VoIP Management and Security (VoIP MaSe 2006), pages 29–34, Vancouver, Canada, April 2006.

[11] H. Abdelnur, R. State, and O. Festor. Advanced Network Fingerprinting. In RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, pages 372–389, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] H. Abdelnur, R. State, and O. Festor. KiF: a stateful SIP fuzzer. In IPTComm '07: Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications, pages 47–56, New York, USA, 2007. ACM.

[14] H. Abdelnur, R. State, and O. Festor. SIPping your Network. In Shmoocon 2008, Washington, USA, February 2008.

[13] H. Abdelnur, R. State, and O. Festor. Fuzzing for vulnerabilities in the VoIP space. In EICAR '08: 17th Annual Conference of the European Institute for Computer Anti-Virus Research, Laval France, 2008.

[7] H. Abdelnur, T. Avanesov, M. Rusinowitch, and R. State. Abusing SIP Authentication. In IAS '08: Proceedings of the 2008 The Fourth International Conference on Information Assurance and Security, pages 237–242, Washington, USA, 2008. IEEE Computer Society.

À ces publications, s'ajoutent la suite logicielle KiF distribuée en logiciel libre ainsi qu'un brevet sur la méthode de fingerprinting.

2 Les services voix sur IP et le protocole SIP

Les infrastructures Voix sur IP comprennent un ensemble d'équipements dédiés (en général orientés vers une application) utilisant des technologies de l'Internet comme transport sous-jacent. Les usagers exploitent des équipements terminaux souvent simples (ex : des téléphones) interagissant avec différents types de serveurs afin de gérer les comptes, la mobilité, la localisation et bien sûr l'établissement d'appel entre usagers. L'établissement d'appel est réalisé sur la base d'un protocole de signalisation dont SIP est devenu un des principaux standards, soutenu notamment par l'IETF. Un nombre croissant d'équipements VoIP embarquent aujourd'hui une pile protocolaire SIP en charge du traitement des messages de ce même protocole. Ces piles implantent un automate complexe. Nous avons retenu le protocole SIP [120] et son instanciation dans le domaine de la voix sur IP pour valider nos modèles. Son architecture est illustrée dans la figure 1.

Celle-ci s'articule autour d'un agent utilisateur (mode client et serveur) et d'un ensemble de serveurs prenant en charge les services de localisation, d'enregistrement, de redirection et de proxy. Ces derniers sont très impliqués dans l'architecture en agissant pour le compte des agents utilisateurs dans de nombreuses interactions.

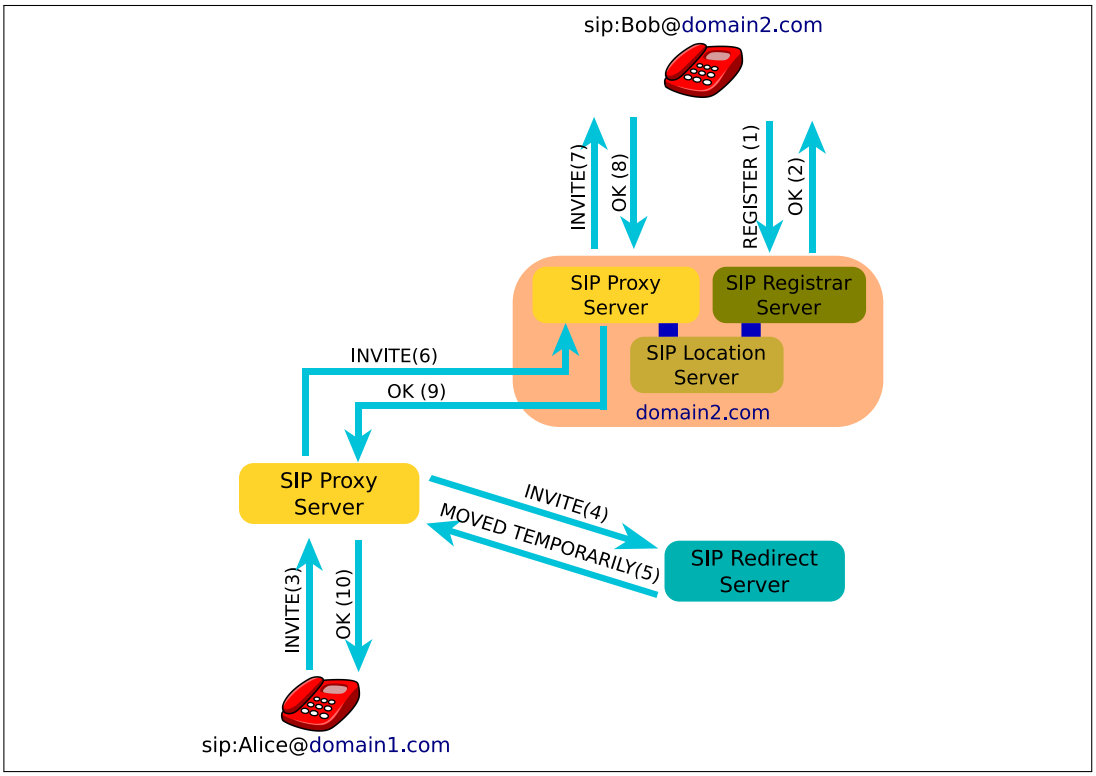


Figure 1: Composants de l'architecture SIP

Le protocole SIP est un protocole en mode texte construit sur la base de protocoles tels HTTP ou SMTP. Les échanges sont structurés en dialogues (relations pair-à-pair entre deux agents) qui incluent des transactions (une interaction requête/réponse). La figure 2 illustre cette hiérarchie.

Les menaces sur les services SIP sont multiples. En s'appuyant sur le modèle de menaces STRIDE défini par Microsoft, elles apparaissent dans six catégories : l'usurpation d'identité, l'altération de données, la répudiation, le déni de service et l'élévation de privilèges. L'alliance VoIPSA qui regroupe des industriels du monde voix sur IP a défini sa propre classification de menaces. Celle-ci inclue : les menaces sociales, l'écoute clandestine, l'interception et la modification de données, la révélation de données, l'abus de services, la dégradation et l'interruption de services par des méthodes physiques ou logicielles.

Un ensemble de bonnes pratiques a été édité pour se prémunir d'un sous-ensemble de ces attaques. Ces pratiques vont de la généralisation de l'usage des mécanismes cryptographiques à la structuration des réseaux et des composants de sécurité attachés tels les pare-feux. Si ces règles sont bonnes, elles ne sont pas suffisantes pour protéger totalement toute infrastructure voix sur IP. Le maillon faible reste ici toujours l'implantation d'un service qui, comme nous le verrons dans la suite de ce chapitre, souffre souvent de multiples vulnérabilités.

3 Architecture d'audit

Offrir à des analystes de sécurité un environnment capable d'automatiser au maximum un ensemble de processus d'audit, nécessite la disponibilité d'une architecture homogène dans laquelle différents outils viennent s'intégrer et sur laquelle le développement de nouvelles applications est possible. Dans le monde de la sécurité voix sur IP, une telle architecture n'existait pas. Celle que nous avons conçue est présentée ci-dessous.

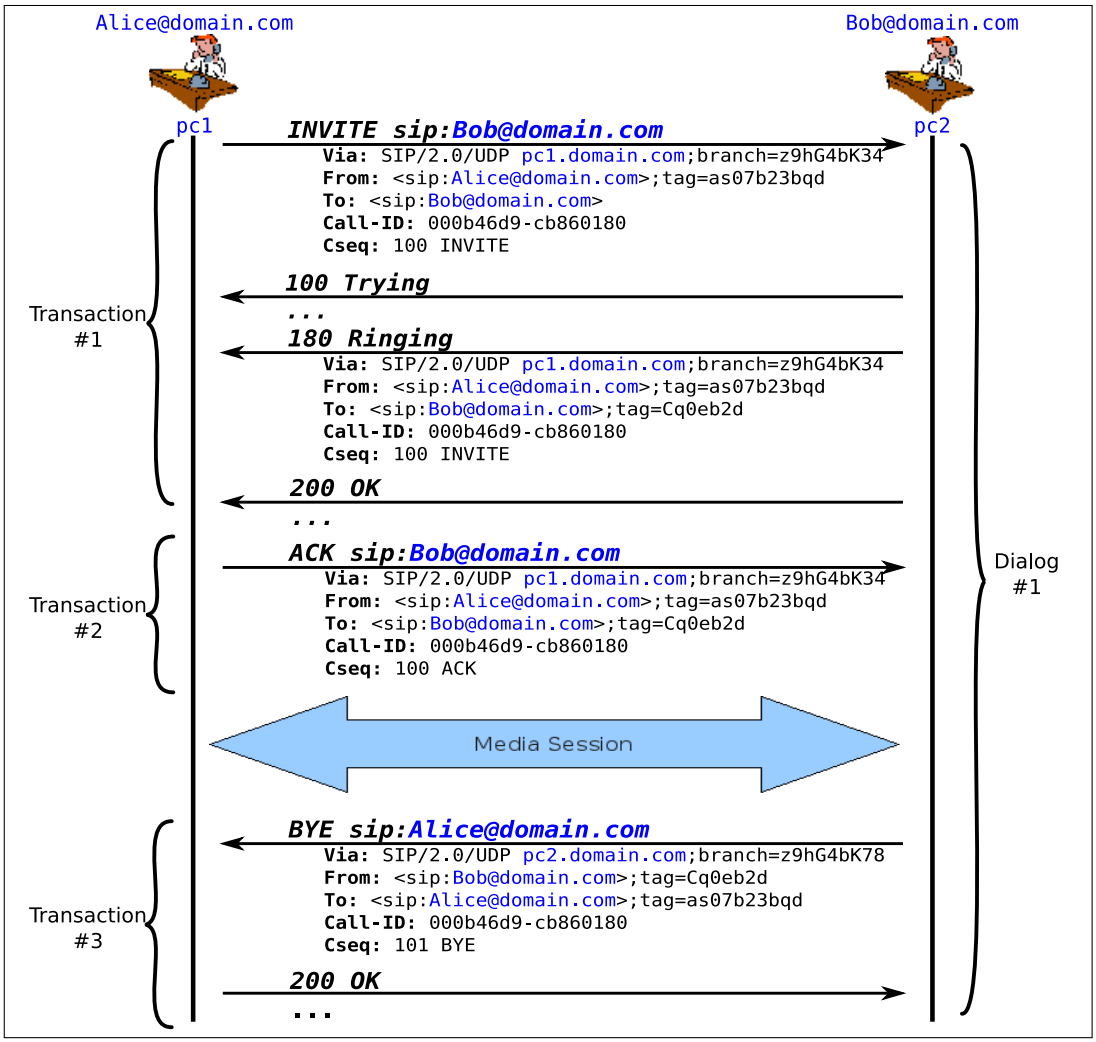


Figure 2: Dialogues et transactions SIP

3.1 Composants de l'architecture

L'architecture que nous definissons est illustrée dans la figure 3.

Elle comporte trois blocs fonctionnels principaux et un modèle d'information intégrateur. Le premier bloc fonctionnel est celui de la découverte des ressources à auditer sur un réseau. Pilotée par un gestionnaire de découverte, ce bloc permet d'intégrer des blocs tiers de découverte tels que des scanneurs passifs et actifs, génériques (SNMP, P0F, ...) ou spécifiques à la couche SIP.

Le second bloc de l'architecture représente les fonctions d'attaque. Autour d'une base d'attaques et de vulnérabilités, il permet d'intégrer des outils et/ou scripts tierces capables d'effectuer les attaques sur les équipements énumérés par le bloc de découverte.

Le dernier bloc représente les applications tiers conçues pour piloter les différents composants de l'architecture. Nous avons développé un tableau de bord dans la mise en œuvre de notre architecture pour illustrer ce bloc.

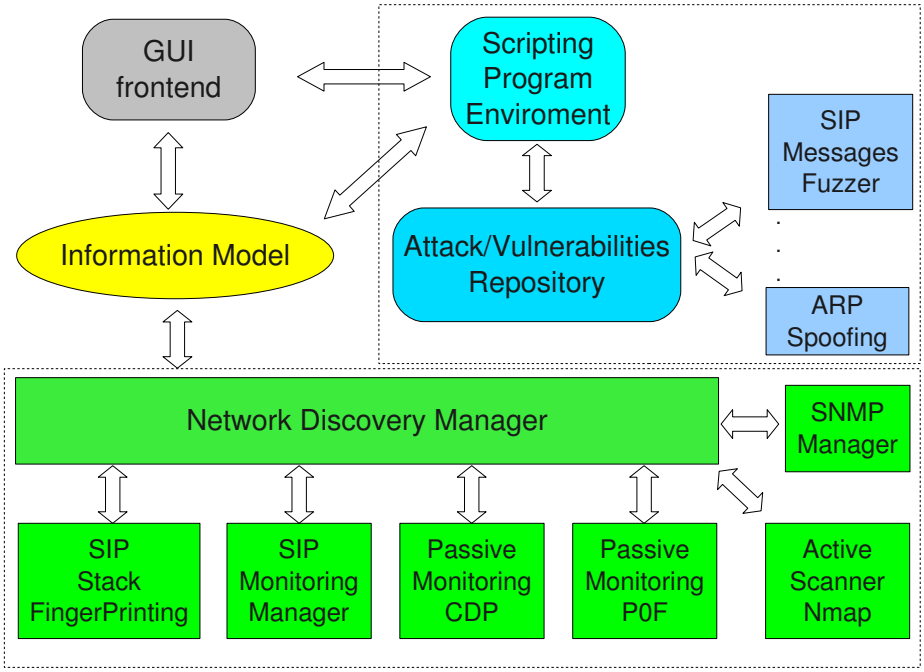


Figure 3: Vue globale de l'architecture d'audit de sécurité VoIP

3.2 Modèle d'information

Le modèle d'information que nous avons conçu est le cœur d'intégration de notre architecture. Il assure la représentation et la cohérence des données entre les services de découverte, les services de fourniture des attaques et les applications tierces. Le modèle est basé sur le standard de modélisation des informations de gestion de réseau CIM (Common Management Information Model) [4] développé au DMTF (Distributed Management Task Force). Notre modèle étend principalement les classes `Device` avec des informations de configuration ainsi que les détails des services opérés par les équipements indentifiés. Le modèle est détaillé dans le chapitre 5 et synthétisé dans la figure 5.2 de ce même chapitre.

3.3 Ecriture de tests d'audit

Une fois que les informations sur la cible ont été collectées et insérées dans le modèle d'information, une phase d'exécution de scripts d'attaque débute. Notre architecture réalise un ensemble d'attaque prédéfinies. Elle permet également à l'analyste de réaliser ses propres scripts d'attaque suivant une modélisation d'arbres d'attaques. La réalisation de ces scripts est faite en langage Python.

3.4 Conclusion

Afin de permettre aux analystes en sécurité de disposer d'un environnement intégré d'analyse, nous avons conçu une architecture unifiée, structurée autour de trois fonctions essentielles et construite sur un modèle d'information riche. Cette architecture à fait l'objet d'une implémentation dans laquelle nous offrons également aux analystes un support pour l'écriture de scripts d'attaques par modélisation d'arbres d'attaques. L'intégralité de l'architecture a été réalisée en Python. Elle n'est pas distribuée à ce jour. Ces travaux ont été publiés dans les actes du 10ème congrès IFIP/IEEE International Symposium on Integrated Network Management [9].

4 Test de vulnérabilités

Le test de vulnérabilités communément appelé Fuzzing est un élément important dans le processus d'évaluation de la sécurité d'un système. Le principe est très simple ; il repose sur l'injection de données aléaltoires dans toutes les interfaces (surface d'attaque) du système testé. La difficulté de l'exercice réside (1) dans la génération de données qui vont permettre de révéler une vulnérabilité dans un équipement cible et (2) dans la construction d'architectures qui vont pouvoir mettre en œuvre de telles fonctions. Le fuzzing vient se placer en complément d'autres approches d'évaluation de la sécurité des logiciels tels que l'audit de code, le reverse engineering, le test actif, la mesure de risques par modélisation d'attaques (graphes, arbres, réseaux d'attaques) et le déploiement de mécanismes de contrôle d'accès et d'inspection de traffic.

Nous avons conçu une nouvelle approche de fuzzing capable, contrairement aux approches existantes sur le marché, d'identifier des failles dans les états avancés d'une interaction protocolaire. Pour cela nous avons développé de nouveaux algorithmes de génération et de suivi de fuzzing couplant apprentissage de protocoles et mutation de données. Nous avons implanté ces algorithmes dans une architecture de fuzzing modulaire et l'avons mise en œuvre sur de multiples équipements. L'approche, l'architecture et les résultats de sa mise en œuvre sont décrits dans ce chapitre.

4.1 Architecture de fuzzing

L'architecture de fuzzing que nous avons développée comporte trois grands blocs fonctionnels. Le premier assure l'échange des messages avec la cible en servant d'interface au point d'accès de service. Le second bloc a en charge les états protocolaires (fuzzer protocolaire) et le troisième gère les données des messages (fuzzer syntaxique). Leurs interactions sont illustrées dans la figure 4.

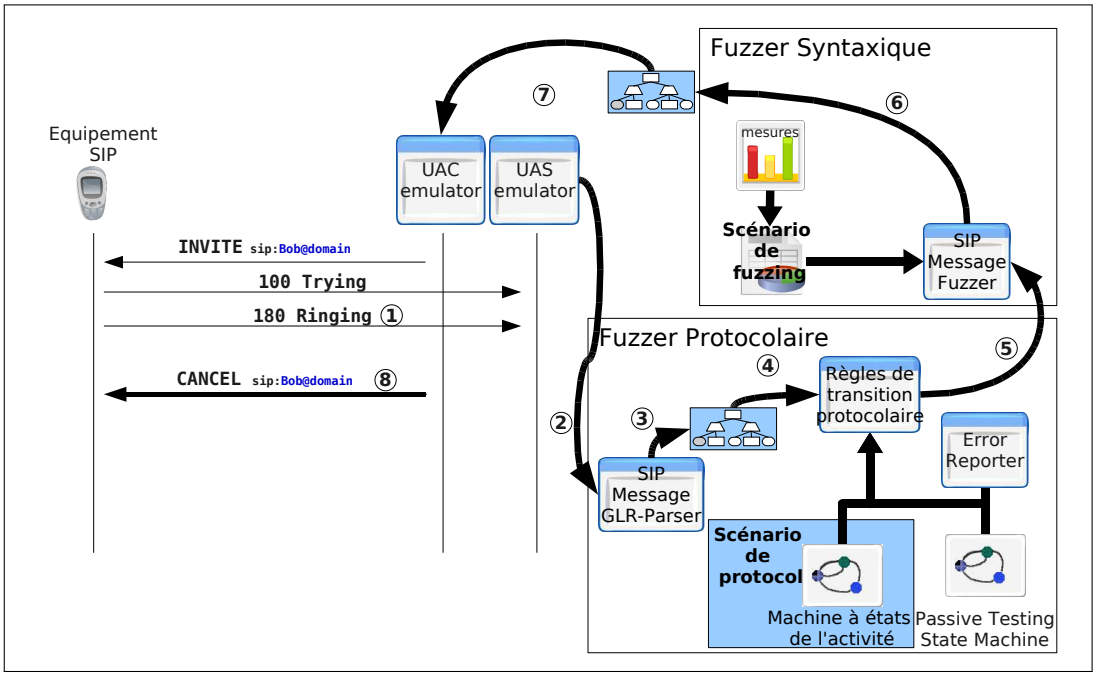


Figure 4: Architecture de fuzzing

Le fuzzer syntaxique a pour objectif unique de générer des messages individuels d'attaque. Il s'appuie pour cela sur la grammaire de ces messages exprimée à l'aide de la métasyntaxe

ABNF (Augmented Backus Naur Form) ainsi que sur un scénario de fuzzing. Ce scénario pilote la génération des règles de production dans la grammaire de la syntaxe. Il peut également dépendre du fuzzer protocolaire afin de générer le message final qui sera envoyé à l'entité cible.

Le fuzzer protocolaire effectue du test passif et actif. Pour cela, deux automates sont requis. Le premier spécifie la machine à états SIP; l'autre spécifie la machine à états de l'activité de test. La première machine est utilisée dans le test passif. Elle contrôle l'occurrence d'un comportement anormal issu de la cible durant la phase de test. Cet automate peut être inféré d'un ensemble de traces SIP relatives à la cible collectées durant des phases opérationnelles normales. Le second automate est utilisé pour du test actif ; il pilote le profil du test de sécurité. Cet automate est défini par l'utilisateur et peut évoluer dans le temps.

4.2 Fuzzing syntaxique

Notre algorithme de fuzzing syntaxique a pour objectif de générer un message fuzzé à destination d'une cible. Pour cela, il prend deux paramètres : la grammaire du protocole et un scénario de fuzzing de syntaxe. Ce dernier comprend les règles à appliquer (nous l'appelons l'évaluateur de fuzzing dans la thèse). Ces concepts sont formalisés au sein d'une grammaire d'expression de fuzzer détaillée dans le chapitre 8, section 8.2.

4.3 Fuzzing protocolaire

La fonction de fuzzing protocolaire est spécifique à un protocole donné car elle requiert une connaissance du comportement normal d'un protocole afin de pouvoir déterminer des états inconsistents d'une implémentation de celui-ci. Dans ce but, notre approche s'appuie à la fois sur du test actif et passif.

Le test passif consiste à surveiller l'intégralité du traffic entre l'attaquant (le fuzzer) et la cible et de le comparer au comportement normal de l'automate d'interaction du protocole. Dans notre approche, cet automate est construit automatiquement par notre approche à partir de l'observation de traçes de la cible en conditions d'utilisation normales (sans attaques). Ces observations nous permettent de construire les automates de base que constituent les transactions dans SIP comme illustré dans la figure 5.

Une alternative à ce mécanisme aurait été de construire l'automate à partir des spécifications du protocole dans les documents normatifs. Ceci n'aurait cependant pas permis de construire les automates adaptés aux équipements cibles car d'une part la norme est très (trop) permissive et d'autre part, nombre d'équipements ne la respectent pas.

Par opposition au test passif qui vise à identifier les états de failles, le test actif consiste à guider l'attaquant ou le fuzzer dans ses échanges avec la cible. Concrètement, le test actif se traduit par des scénarios qui peuvent être soit implémentés à la main par un testeur, soit générés automatiquement. Dans l'environnement actuel, nous utilisons le modèle d'automates finis étendus pilotés par des événements (EEFSM) défini par Lee et al. [90]. Un test actif est centré sur un dialogue et peut intégrer plusieurs transactions. Tout nœud de l'automate comprend les éléments suivants :

- la nature, le type et la direction du message attendu,
- la pré-condition à satisfaire dans l'environnement de l'automate,
- la fonction à appliquer sur la transition (scénairo de traitement de la syntaxe),
- des contraintes temporelles,
- un poids permettant la sélection de transitions concurrentes.

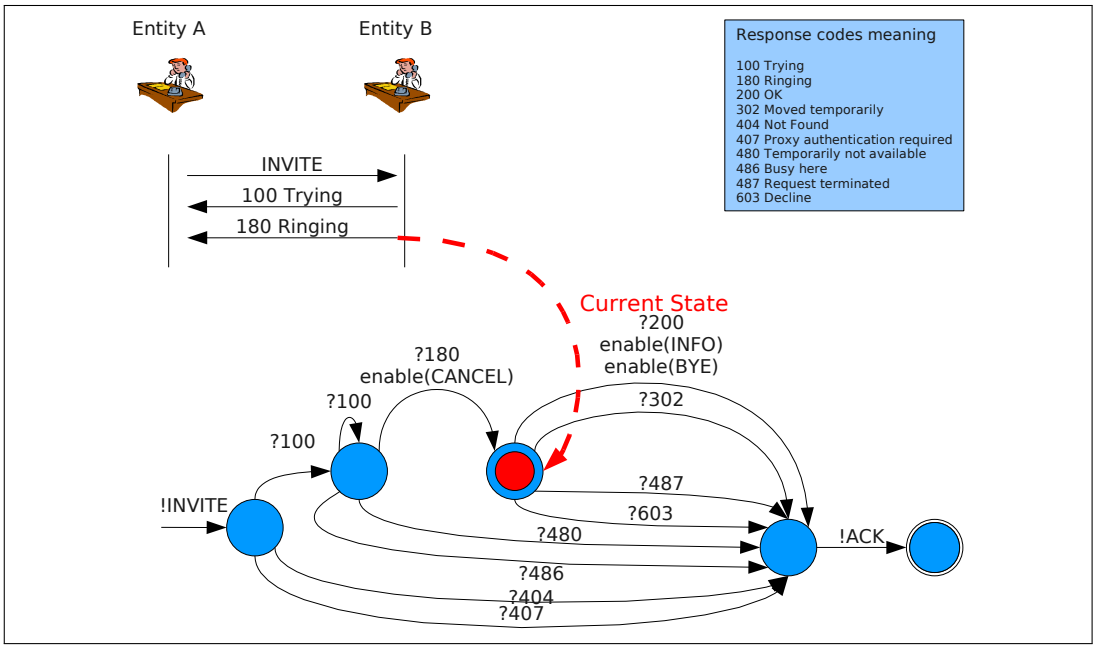


Figure 5: Automate de fonctionnement appris à partir d'un message INVITE
Les transitions marquées (?) indiquent que les messages correspondent à l'entité qui répond. Le marquage (!) indique que les messages proviennent de l'entité cliente.

Un scénario de test est illustré dans la figure 6.

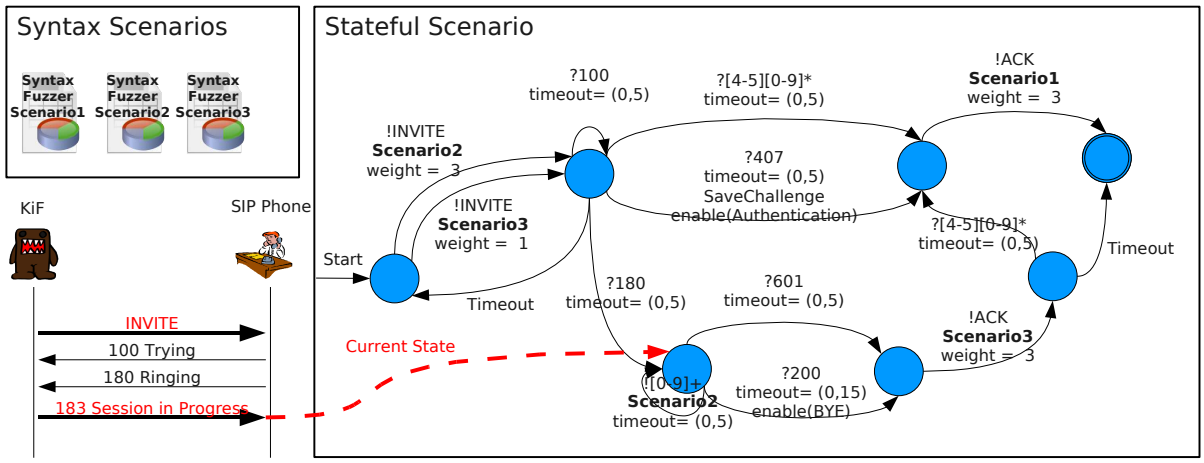


Figure 6: Exemple de scenario de test SIP

Il est possible de définir de multiples scénarios de test dans le fuzzer.

4.4 Evaluation

L'application de notre approche à un grand nombre d'équipements voix sur IP a permis de découvrir de multiples vulnérabilités. A ce jour, aucun équipement testé ne s'est montré infaillible. Dans cette section, nous synthétisons les grandes familles de vulnérabilités identifiées.

Faiblesses dans la validation des entrées

La vulnérabilité que nous avons rencontrée le plus fréquemment est liée à un filtrage extrêmement faible (voir inexistant) des données fournies en entrée d'entités voix sur IP via le canal SIP. Ce filtrage, lorsqu'il existe, ne traite pas proprement les méta caractères, les caractères spéciaux, les données de grande longueur ou les caractères spécifiques de formatage. Les failles qui en résultent sont dues à des débordements de tampon/tas ou des vulnérabilités de type "format string". La cause la plus probable à cela est que les développeurs de ces systèmes sont partis d'un modèle de menaces dans lequel la signalisation SIP est supposée générée par des piles protocolaires saines et éprouvées. Une raison plus simple encore peut être l'absence dans les processus de conception de certains de ces équipements de toute ou partie de la dimension sécurité. Le véritable danger de cette vulnérabilité provient du fait que dans la grande majorité des cas, un très faible nombre de paquets peut littéralement paralyser un réseau VoIP complet. Ceci est d'autant plus dangereux que dans le cas présent, les messages SIP sont transportés sur UDP, ouvrant la porte à des attaques efficaces effectuées de façon furtive par des techniques simples de spoofing IP. Nous mettons en exergue deux cas extrêmes de vulnérabilités découvertes par notre approche : la première vulnérabilité (publiée dans CVE-2007-4753) révèle que dans le cas étudié, même le test le plus simple de vérification de l'existence de données en entrée n'est pas effectué. Cette absence de vérification permet des attaques extrêmement simples et efficaces telles que l'envoi d'un paquet vide. Le second cas (CVE-2007-1561) est situé à l'extrême du premier sur l'échelle de la complexité. Ici un serveur VoIP est vulnérable à une attaque dont la structure de données d'entrée est relativement complexe. Le danger repose dans ce cas sur le fait qu'un unique paquet va détruire le serveur voix sur IP de cœur et ainsi rendre indisponible l'ensemble du service VoIP associé. Se prémunir de telles attaques à un niveau de défense réseau est possible via des techniques d'inspection profonde de paquets couplées à des équipements de filtrage de paquets spécifiques au domaine.

Vulnérabilités de suivi protocolaire

Les vulnérabilités de suivi protocolaire vont au delà du simple filtrage d'un unique message SIP. Dans ce type de vulnérabilités, plusieurs messages vont amener un équipement cible dans un état inconsistant ; tout message utilisé dans cette chaîne d'attaque considéré en isolation ne violera pas la spécification normative du protocole SIP. Ces vulnérabilités proviennent en grande majorité d'une faiblesse dans l'implémentation des automates du protocole. Elles peuvent être exploitées de trois façons différentes :

- L'équipement peut recevoir des entrées qui ne sont pas attendues dans l'état courant du protocole : par exemple en envoyant au système un BYE alors qu'il s'attend à recevoir un INVITE,
- L'entrée peut prendre la forme de messages simultanés dirigés vers plusieurs états du protocole,
- De faibles variations dans les champs de suivi de dialogues et/ou transaction SIP peuvent amener un équipement vers un état inconsistant.

La découverte de telles vulnérabilités est un problème difficile. Le processus de fuzzing doit ici être capable d'identifier où et à quel moment un équipement cible ne suit pas rigoureusement le protocole et quels champs des messages peuvent être "fuzzés" pour révéler la vulnérabilité. L'espace de recherche est dans ce cas gigantesque, couvrant de multiples messages et champs de données ; l'utilisation de techniques de fuzzing avancées pilotées par des méthodes d'apprentissage

est ici indispensable. Comme pour le cas précédent (vulnérabilités liées au filtrage des données), les vulnérabilités révélées par l'application de notre méthode sont de complexité variable.

Un cas simple est celui de la (CVE-2007-6371). Ici, l'envoi prématuré d'un message CANCEL peut amener l'équipement dans un état inconsistant qui aboutit à un déni de service. Le danger majeur de ce type d'attaques est qu'à ce jour, aucun pare-feu applicatif ne peut suivre et inspecter un si grand nombre de flux et que même dans le cas où les signatures sont connues, des versions polymorphiques d'attaques efficaces peuvent aisément être obtenues et ainsi passer entre les mailles des systèmes de protection. A ce jour malheureusement, aucune solution efficace pour la prévention de ce type d'attaque n'existe.

Nous avons également identifié un nombre conséquent de vulnérabilités liées à des faiblesses dans les implémentations. Ces faiblesses concernent des implémentations cryptographiques faibles (CVE-2007-5468 et CVE-2007-5469), des supports d'injection SQL et/ou Javascript permettant de la fraude à la facturation et la capacité de certains téléphones à permettre des écoutes distantes sans aucune action du destinataire (CVE-2007-4498).

Vulnérabilités dans la spécification du protocole

Nous avons consacré une part importante de notre activité à la recherche de vulnérabilités sur des implémentations spécifiques du protocole SIP sans initialement considérer la sécurité du protocole en soi. C'est lors de l'exécution d'un scénario de fuzzing complexe qui nous avons relevé la même anomalie (et vulnérabilité apparente) sur tous les équipements sous test. Ceci nous à naturellement conduit à lancer une analyse sur la spécification du protocole SIP, notamment en utilisant des techniques formelles et outils supports tels AVISPA . Cette analyse nous a permis d'identifier la vulnérabilité dans la conception même du protocole, vulnérabilité qui rend toute attaque d'escroquerie à la facturation possible sur tout réseau voix sur IP.

Le problème vient du fait qu'une attaque classique de type relais est possible en forçant une entité appelée à émettre un message de type RE-INVITE. Cette attaque étant nouvelle, générique et sévère, elle est naturellement dangereuse. Voici comment elle se matérialise : un attaquant établit un appel avec sa victime. Sa victime répond (décroche) et est amenée à mettre l'appelant en attente (il existe plusieurs méthodes pour la conduire à entreprendre cette action, la plus simple étant qu'un complice appelle la victime alors que celle-ci est en communication avec l'attaquant). Lorsque l'attaquant reçoit le message SIP re-invite qui spécifie la mise en attente, celui-ci peut demander à la victime de s'authentifier. Cette dernière authentification peut être utilisée par l'attaquant pour se substituer à la victime sur son propre proxy et passer ainsi des appels frauduleux à l'insu de la victime.

4.5 Synthèse

Il existe un grand nombre d'outils de fuzzing sur le marché. Tous génèrent de façon plus ou moins intelligente les données à injecter dans la cible afin de la perturber. Notre principale contribution sur ce domaine est d'avoir conçu un modèle de fuzzing qui peut aller tester une cible dans des états protocolaires avancés, ce qu'aucun autre fuzzer ne fait à ce jour.

Nous avons implanté notre méthode de fuzzing et l'avons instanciée sur SIP. Les résultats sont très encourageants avec un ensemble important de vulnérabilités identifiées sur tous les équipements testés. L'outil KiF qui réalise l'architecture de fuzzing est distribué sous license Open Source par l'équipe.

Ces travaux ont été publiés dans les actes de 1^{st} international conference on principles, systems and applications of IP Telecommunications, IPTComm 2007 [10] et l'outil KiF a été présenté à Shmookon 2008 [14].

5 Fingerprinting

5.1 Contexte

L'appelation *fingerprinting* regroupe en informatique communicante, l'ensemble des techniques qui permettent d'identifier une entité distante (un composant physique, un pilote de périphérique, un système d'explotation, un service ou une application) par l'empreinte que celle-ci génère en échangeant des messages sur un réseau informatique. Comer et Lin [54] ont, dans le milieu des années 1990 contribué à populariser cette technique qui s'est fortement développée depuis et qui trouve de nombreux domaines d'applications en gestion de réseaux (gestion de l'inventaire par exemple) et en sécurité.

Il existe aujourd'hui deux grandes familles d'approches qui permettent de réaliser des services de fingerprinting : les approches actives et les approches passives. Le fingerprinting actif utilise l'injection de messages pour déclencher des réactions spécifiques chez sa cible. Ces réactions (en général des réponses à l'injection), couplées aux injections sont utilisées pour identifier la source. Cette identification est réalisée par comparaison de signatures de couples injection/réponse à des couples similaires dans une base de connaissance comportant des triplets (injection/réponse/identification de cible). La difficulté essentielle de cette approche est la construction de la base de connaissances. Ces approches sont en général extrêmement précises et donnent d'excellents taux de réussite. Elles ont cependant le désavantage d'être invasives en générant un traffic supplémentaire perturbateur sur le réseau.

L'approche passive se limite elle à observer du traffic standard sur un réseau et ne s'appuie que sur ces données pour établir une identification. Naturellement non intrusive, elle donne en général de moins bon résultats que sa contrepartie active.

5.2 Contributions

Nous proposons une nouvelle méthode de fingerprinting passif basée sur l'exploitation de la structure des messages plutôt que de son contenu, contenu trop facilement modifiable pour leurrer les algorithmes de fingerprinting existants.

L'ensemble des éléments qui interviennent dans notre approche de fingerprinting ainsi que leur enchaînement sont présentés dans la figure 7.

Notre approche nécessite la connaissance préalable de la grammaire du protocole (de ses messages). Celle-ci nous sert de surface d'observation. Sur la base de la grammaire, nous effectuons une inférence structurelle d'un ensemble de messages collectés d'un équipement cible (le terminal pour lequel nous cherchons à construire une base de signatures structurelles). Ceci est réalisé dans une phase dite d'apprentissage ou de découverte de signatures. Deux tâches composent cette phase.

La première tâche consiste à identifier des champs variables (appelés variants dans notre approche) de la grammaire du protocole, pour un équipement donné. Concrètement, cela revient à analyser un grand nombre de traces contenant des messages issus d'un équipement (requêtes et/ou réponses). Chaque message est traduit en un ensemble de nœuds (identifiés par chemins dans la grammaire) et ces identités de nœuds sont comparées à toutes celles d'autres messages du même équipement. Le choix des nœuds à comparer se base sur un algorithme de calcul des ressemblances tel que défini par A. Broder dans [34]. Le résultat de cette première phase est un ensemble de nœuds invariants tirés de l'analyse de tous les nœuds observés dans l'ensemble des messages d'apprentissage (traces SIP d'un équipement donné).

La seconde tâche a pour objectif d'extraire d'un ensemble de signatures de différents équipements cette fois-ci, celles qui vont permettre de les distinguer (nous appelons ces signatures des caractéristiques).

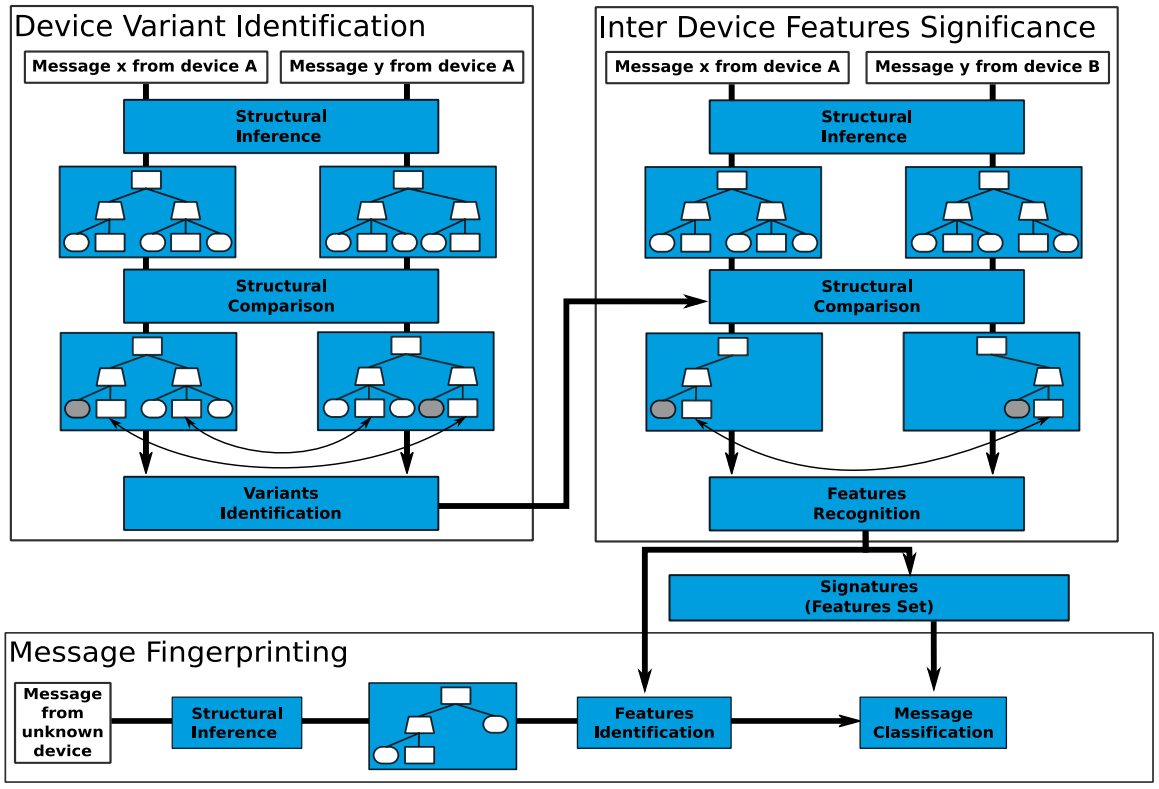


Figure 7: Blocs fonctionnels de l'architecture de fingerprinting

Partant d'un ensemble de traces de n équipements appris, nous recherchons au sein des invariants de chaque équipement ceux qui sont uniques à un équipement donné et qui vont permettre de le distinguer de manière unique. Ceci donnera dans le cas idéal un ensemble de caractéristiques pour tous les équipements. Une fois cette base établie, il suffit lors de l'observation d'un message sur le réseau de rechercher quels sont ses invariants et quel équipements ils caractérisent.

Le principe est celui-décrit ci-dessus. L'implémentation réalisée opère de façon complémentaire. La première phase calcule tous les variants d'un équipement. La seconde opère non pas sur les invariants des équipements mais sur leurs variants fournis et compare les messages de traces de tous les équipements pour en extraire les invariants intra-équipement et surtout ceux d'entre eux qui vont être des variants inter-équipements et former les caractéristiques.

5.3 Evaluation

Nous avons implémenté le modèle de fingerprinting et l'avons testé sur des traces réseau réelles (21981 messages issus de 26 équipements différents). L'apprentissage a été réalisé sur 15 % des traces. Les résultats de classification de l'ensemble des 21981 messages sont donnés dans la table 1.

Ces résultats démontrent la qualité de l'algorithme. L'analyse des faux négatifs montre que ceux-ci proviennent essentiellement de deux primitives de services qui véhiculent trop peut de données pour être classifiées correctement. Ces messages sont des messages d'options, de réponse 100 et d'ACK. L'analyse du passage à l'échelle de la phase d'apprentissage démontre que l'approche reste très efficace avec des taux d'apprentissage de traffic de l'ordre de 10%.

Classification	Vrai Positif 21422	Faux Positif 32
	Faux Négatif 490	Vrai Négatif N.A.
Justesse 0.998	**Sensibilité** 0.976	**Spécificité** 0.999

Table 1: Résultats obtenus avec notre méthode de fingerprinting

5.4 Synthèse

Etre capable d'identifier de façon explicite l'ensemble des entités qui opèrent dans une infrastructure est essentiel à toute opération d'audit de sécurité. Nous avons pour cela conçu une nouvelle méthode passive exploitant les propriétés structurelles des messages échangés sur un réseau. Notre approche est automatique et son évaluation a démontré son efficacité dans une infrastructure Voix sur IP intégrant un nombre conséquent de terminaux hétérogènes.

Ces travaux ont fait l'objet d'un publication au 11ème Symposium RAID (Recent Advances in Intrusion Detection) [11] et ont fait l'objet d'un dépôt de brevet international en 2007. L'impléméntation du modèle est opérationnelle et son extension se poursuit au sein de l'équipe.

6 Conclusion

6.1 Synthèse des contributions

Nos travaux ont porté sur l'analyse de sécurité de services dans l'Internet. L'application cible retenue est celle de la voix sur IP utilisant la signalisation SIP. Notre objectif était de concevoir une architecture d'analyse de sécurité qui permette à la fois de construire l'information nécessaire à cette fonction et d'effectuer des services spécifiques tels que le fuzzing et le fingerprinting. Ces travaux ont mené à trois contributions principales : (1) l'architecture d'audit, (2) un nouveau procédé de fingerprinting et (3) un modèle de fuzzing générique. Nos contributions, résumées ci-dessous ont toutes été implantées et validées sur des infrastructures VoIP.

Nous avons conçu une architecture d'audit de sécurité pour la voix sur IP intégrant des informations issues de multiples sources (SNMP, fingerprinting, NMAP, ...). Un modèle d'information dédié permet cette intégration. Ce modèle, ainsi que les différents blocs fonctionnels de l'architecture, ont été implantés dans un prototype en Python et expérimentés sur la plateforme VoIP de l'équipe.

Nous avons conçu une nouvelle méthode pour générer des systèmes de fingerprinting basés sur l'analyse structurelle des messages d'un protocole. Notre solution permet d'automatiser ce processus lourd, manuel dans la grande majorité des approches concurrentes. Le modèle et ses composants fonctionnels ont fait l'objet d'un dépôt de brevet. Tout comme l'architecture d'audit, le service de fingerprinting implantant notre méthode a été réalisé en Python et testé sur plusieurs instances de réseaux VoIP. Les évaluations montrent d'excellents résultats.

Nous avons élaboré un modèle de fuzzing à états. Ce modèle flexible permet de découvrir des vulnérabilités dans des états avancés d'une réalisation d'un protocole. Son implantation et son application au protocole SIP ont démontré d'une part l'intérêt du modèle et d'autre part la limite des approches concurrentes incapables de découvrir des vulnérabilités au delà de la première primitive de service d'un protocole donné. Nous avons identifié de multiples vulnérabiltés dans toutes les implantation et équipements testés et avons démontré la réalisabilité d'un nombre d'attaques ignorées par les standards et consortiums autour de la voix sur IP. Finalement, nous

avons à l'aide de notre fuzzer indentifé une faille dans la spécification du protocole SIP, faille que nous avons validée avec nos collègues de la réécriture et pour laquelle nous avons élaboré une contre-mesure.

6.2 Travaux futurs

Notre approche de fingerprinting s'est montrée très efficace sur le protocol SIP qui a la caractéristique d'être très bavard. L'approche que nous avons développée s'appuie uniquement sur la structure des messages pris en isolation. Un extension naturelle de l'approche, en plus de son application à d'autres protocoles bien sûr, portera sur l'exploitation de dépendances intra- et inter-messages pour optimiser le fingerprinting. Les dépendances intra-message portent sur des corrélations entre différents champs au sein d'un même message. Une mesure de l'entropie peut être utile ici et peut servir de base d'analyse pour la sélection de signatures. La même approche peut probablement s'appliquer à l'échelle d'une session.

Sur le domaine du fuzzing, nous comptons étendre à court terme l'environnement au travers du support de protocoles cibles suppélmentaires. Ceci permettra d'une part de démontrer concrètement la généricité de nos méthodes et d'autre part d'offrir à la communauté un spectre de tests plus large. Une seconde piste que nous explorons actuellement porte sur la conception de mécanismes permettant d'évaluer la qualité et la couverture d'un processus de fuzzing et/ou de prendre en compte des informations provenant de la cible pour guider le processus de fuzzing. Pour ces deux objectifs, un mécanisme de mesure d'impact et de collecte d'informations sur la cible est indispensable. Nous avons démarré la réalisation d'un tel service dans un environnement virtualisé (Xen). D'autres sources de données tels que les journaux sont également intéressantes. Sur ce dernier point nous coopérons actuellement avec des chercheurs d'Alcatel-Lucent Bellabs sur la conception d'un format unifé de journaux pour de la signalisation SIP. Au dela de la collecte d'information, nous envisageons la conception d'algorithmes d'adaptation du fuzzing capables de diriger les tests et le cas échéant de décider de l'arrêt de ceux-ci en fonction d'une couverture cible.

L'autonomie de nos approches de fingerprinting et de fuzzing est un objectif à plus long terme. Nous entendons par autonomie, la capacité de nos approches à s'adapter automatiquement sans intervention humaine à de nouveaux protocoles. Ceci est intéressant dans des cas ou la spécification du protocole n'est pas disponible ou trop complexe (voire insuffisamment précise) pour être exploitable. Nous pensons qu'il est intéressant de coupler des systèmes de reverse engineering (par exemple de reconstruction de protocole à partir d'analyse de traces) avec des systèmes de fuzzing et de fingerprinting. Nous focalisons dans un premier temps sur un sous-ensemble de protocoles avec comme but de dériver des fuzzers de façon automatique.

Dans une architecture d'analyse de sécurité d'une infrastructure, il y a de nombreux autres sources d'informations qui peuvent être extrêmement utiles. Les configurations de pare-feux en est une particulièrement intéressante qui peut aider soit à construire des fuzzers pour ces derniers soit à guider les processus de fuzzing pour atteindre les équipements cibles placés derrière ces pare-feux. Nous avons démarré une activité sur ce dernier point dans le cadre de la coopération avec Alcatel-Lucent Bellabs.

General Introduction

Voice over IP (VoIP) is emerging as the key technology of the Internet. VoIP networks are in a major deployment phase and are becoming widely accepted due to their extended functionality and cost efficiency. With the recent evolution in the VoIP market, where more and more devices and services are being pushed on a very promising market, assuring their security becomes crucial.

Meanwhile, as VoIP traffic is transported over the Internet, it is the target of a range of attacks that can jeopardize its proper functionality. Among the most dangerous threats to VoIP, failures and bugs in the software implementation will be high on the list of vulnerabilities.

Summary of Contributions

The first contribution of this thesis is a VoIP specific security assessment framework. Assessment is here automated with integrated discovery actions, data management and security attacks allowing to perform VoIP specific penetration tests. These tests are important because they permit to search and detect existing vulnerabilities or misconfigured devices and services. This contribution consist in an elaborated network information model usable for VoIP assessment, an extensible assessment architecture and its implementation, as well as in a comprehensive framework for defining and composing VoIP specific attacks.

Security assessment tasks and intrusion detection systems do rely on automated fingerprinting of devices and services. Most current fingerprinting approaches use a signature matching scheme, where a set of signatures is compared to traffic issued by an unknown entity. The entity is identified by finding the closest match with the stored signatures. These fingerprinting signatures are found mostly manually, requiring a laborious activity and needing advanced domain specific expertise. The second contribution of this thesis describes a novel approach to automate this process building a flexible and efficient fingerprinting systems able to identify the source entity of messages in the network. A passive approach is followed without interacting with the tested device. Application level traffic is captured passively and inherent structural features are used for the classification process. A new technique is described and assessed for the automated extraction of protocol fingerprints based on arborescent features extracted from an underlying grammar. The technique has been successfully applied to the Session Initiation Protocol (SIP) used in Voice over IP signalling.

The third contribution addresses the issue of detecting vulnerabilities using a stateful fuzzer. An automated attack approach capable of tracking the state context of a target device is described. The approach has been implemented and was able to discover vulnerabilities in market leading equipments and software. Practical experience gained over a two years period in searching for vulnerabilities in the VoIP space is described and illustrated. A landscape of dangerous vulnerabilities capable to lead to a complete compromise of an internal network is shown. All of the described vulnerabilities have been disclosed responsibly by the Madynes group and they were discovered using the developed fuzzing software KiF. However, this manuscript also describes mitigation techniques for all described vulnerabilities.

Organization of the Thesis

This manuscript is organized in four parts. The first part describes the current state of the art of VoIP security assessment. Specifically, chapter 1 contains a general introduction to Voice over IP networks, their difference with the traditional telephony, risks and best practices along with a description of the SIP protocol which is the preferred used case of the work done in this thesis. Chapter 2 describes the different assessment methodologies, current state of deployment of assessment tools; threat modelling is also covered here. Chapter 3 describes the evolution of fingerprinting research, the different classification of network fingerprinting and which are the usage and target of fingerprinting approaches. Chapter 4 concludes the state of the art part by focusing on system vulnerability checking. It makes special emphasis on an emerging discipline of fuzzing in the assessment community. The chapter describes the typical mistakes conceived in vulnerable softwares, how fuzzing methodologies behave over the softwares to trigger possible problems and the different approaches conceived by this discipline.

The second part of the manuscript describes the first contribution. This is an assessment approach which exploits different existing tool in order to build a common information model for VoIP assessment (specifically to SIP). Thus, information gathered in the model can later be used to conduct testing attacks in order to evaluate the network security level.

The third part of the thesis describes a novel fingerprinting approach. It describes how messages are represented by the system, how the system making use of such representation can automatically discriminate signatures out of traces and finally the experimental results are shown.

The fourth part describes our fuzzing technique, how syntax and state are built and explored in order to perform an extensive testing. Experimental results are described showing the most remarkable vulnerabilities that we found. Indeed, a vulnerability has been found which is reported to be a flaw in the design of the SIP protocol; this flaw is described and a mitigation solution is given.

Finally this thesis summarizes a general conclusion and with the envisioned future work.

Part I

Background

Chapter 1

VoIP Services

From more than a century, the telephone system has been deployed with the purpose of allowing remote entities to communicate over a switched infrastructure. This technology has evolved from providing services for just a few interconnected private telephones to a huge public telecommunication network.

Initially this technology allowed to transmit audio between two telephones only if both locations were physically connected by a single dedicated cable. As telecommunications were becoming popular, this approach demonstrated to be highly inefficient and not scalable. Therefore, a new architecture component has been conceived to interconnect two different users from a group of telephones, the *telephone operator* or *switch*. This component was connected directly to a set of telephones and served any request between its users to place them in a call with someone else. Obviously, as the use of the service gain interest, a hierarchical structure of switches has been designed to interconnect different cities, states and countries. This architecture is known as the Public Switched Telephone Network (PSTN).

The PSTN consists of three major components as illustrated in Figure 1.1 and described below:

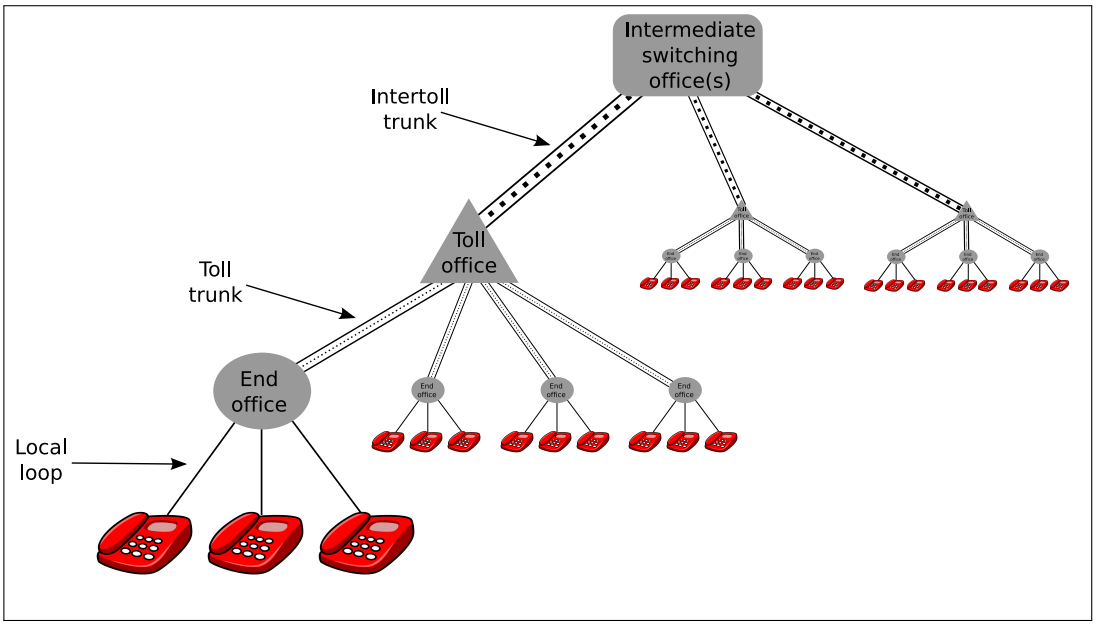


Figure 1.1: Public Switched Telephone Network architecture

Local loop is the physical cable which connects a subscriber's telephone to an End office switch.

Trunks are the lines used to connect between different switching offices. These lines support multiple calls at the same time. Depending on their location in the hierarchy, they are called either *toll connecting trunk* or *interoffice trunks*.

Switching Offices are the components in charge of accounting and routing incoming calls. These components are known as *end offices, toll offices* or *intermediate switching offices*.

From a functionality point of view, there are two main factors that play against the traditional telephony, opening doors to new telecommunication technologies [49]. First, the intelligence of the PSTN network resides in the core components, the *switching offices*. Whenever a telephone user initiates a call, receives or generates any type of request, the end office switch has been generating all those requests on his behalf. Therefore, new functionalities have to be implemented in the switches; making the equipment more expensive and difficult to cope with all the features needed by the set of subscribed users. Secondly, the PSTN is a circuit-switched network where for each routing call, it needs to allocate a fixed rate of bandwidth; 64-kbps. Even if the bandwidth is used or not, the system can not reuse such resources for other purposes and finally it must charge the parties for such consumption.

Voice over IP (VoIP) is being considered as the future for the telecommunications. VoIP infrastructures instead, are built using application level devices exploiting IP technology as the underlying transport layer. IP networking is founded on a packet-switching technology where bandwidth is used as it is needed. Thus, making the service more cost-efficient and extensible (e.g. since the bandwidth rate is not fixed, then it may allow new features like video). Secondly, an important difference is that in the VoIP networks, the intelligence is distributed over the devices rather than just in the core switching equipment as it is in the PSTN. Thus, as services are not exclusive to the operators, it allows competition and directly decreases the production cost.

To end users the migration process is transparent, they operate simple end devices (phones) by leveraging different types of servers in order to manage the mobility, localization and user to user call establishment. This call establishment is performed by signaling protocols, where the Session Initiation Protocol [120] (SIP) is becoming the de facto standard body endorsed protocol. A description of the SIP protocol is given below in section 1.1.

VoIP services have recently become widely deployed mostly because of their lower cost comparing to the traditional telephony, their facility to integrate new services like video conferences, instant messaging, presence services, etc. Although, porting telecommunication to a public data network may introduce some drawbacks comparing it with the traditional telephony, for instance:

- the current availability provided in VoIP is still lower than in traditional telephony. It is expected that in the near future the two services will become equal in terms of expected availability.
- As in VoIP networks the bandwidth is not allocated at initiation of a call, Quality of Service (QoS) is a challenge to be addressed since it has to deal with *available bandwidth, latency, packet loss, jitter*, etc.
- Since in VoIP all the data is supposed to be sent over public networks such as the Internet (in practice some VoIP service providers do use dedicated trunks), security issues must be addressed.

The PSTN is a closed trusted neighbor network where security is achieve by well-defined network boundaries [56]. Since the terminals (e.g. the telephone lines) are dumb entities, the

manipulation of the system is only possible if physical access to the core network is granted [133]. First, it requires expensive material and secondly it requires break in the facilities of the telecommunication operators. However, before the adoption of the Signalling System #7 Protocol [3] (SS7), the network was exposed to different attacks and frauds where having access to the core equipments was not required. Previous signalling protocols were managing the call-setup signaling in-band, e.g. using multi-frequency tones. Thus, they allowed attackers to send such frequencies during a call to achieve unauthorized task by entering in the operator mode of the trunks, such as making free phone calls [119].

VoIP relies on networks in which intelligence is distributed. Access to the network is granted by just having one end device connected to it. Then, for voice data communications, security becomes an important issue to be addressed. Section 1.2 refers to the possible attacks encountered in VoIP networks, mitigation and best practices.

1.1 Session Initiation Protocol (SIP)

The SIP [120] protocol emerged as a standard from the the Internet Engineering Task Force[1] (IETF) under the document reference RFC 2543, and it evolved over time (now known as RFC 3261). The Internet Assigned Numbers Authority[2] (IANA) shows more than 40 extensions that have been standardized.

The designers of SIP leveraged well proven concepts from the HyperText Transfer Protocol [60] (HTTP) to build a robust and multi-features signaling protocol. The advance of highly dynamic services deployed over multimedia enabled networks reaching powerful end user equipment had to be matched by an appropriate signaling protocol. SIP is a signalling protocol used to initiate, manage and tear down sessions. SIP is located at the application layer of the TCP/IP model [33] and it has been designed to be independent of the underlying transport layers. The media session is usually managed by the Real-time Transport Protocol [123] (RTP) encapsulating encoded audio/videodata. The specific characteristics of the RTP session are negotiated in the SIP session. In the simplest case, call establishment with SIP has to be able to let the two communicating partners send RTP data between their two locations.

1.1.1 Architecture

SIP is a decentralized protocol, where the intelligence is distributed through the entities that the network is composed of. Thus, different components have been defined in the SIP architecture playing different roles in a deployed network. Figure 1.2 illustrates these components and each one of them is described below:

- **User Agents (UA):** they are the end-user devices in a SIP network. A regular UA can be a soft or hard phone (i.e. software or hardware endpoints) as well as a gateway that connects to other VoIP protocols or for instance to the PSTN. Each UA can be divided into two logical entities:
  - *User Agent Client (UAC)* which is the one in charge of initiating the requests.
  - *User Agent Server (UAS)* which is the one responsible for generating the responses to the received requests.
- **SIP Location Sever:** is referred to as a storage component. This server is used to keep a database containing current location addresses, features and other preferences of all the

[1] `http://www.ietf.org` last checked December 2008
[2] `http://www.iana.org/assignments/sip-parameters` last checked December 2008

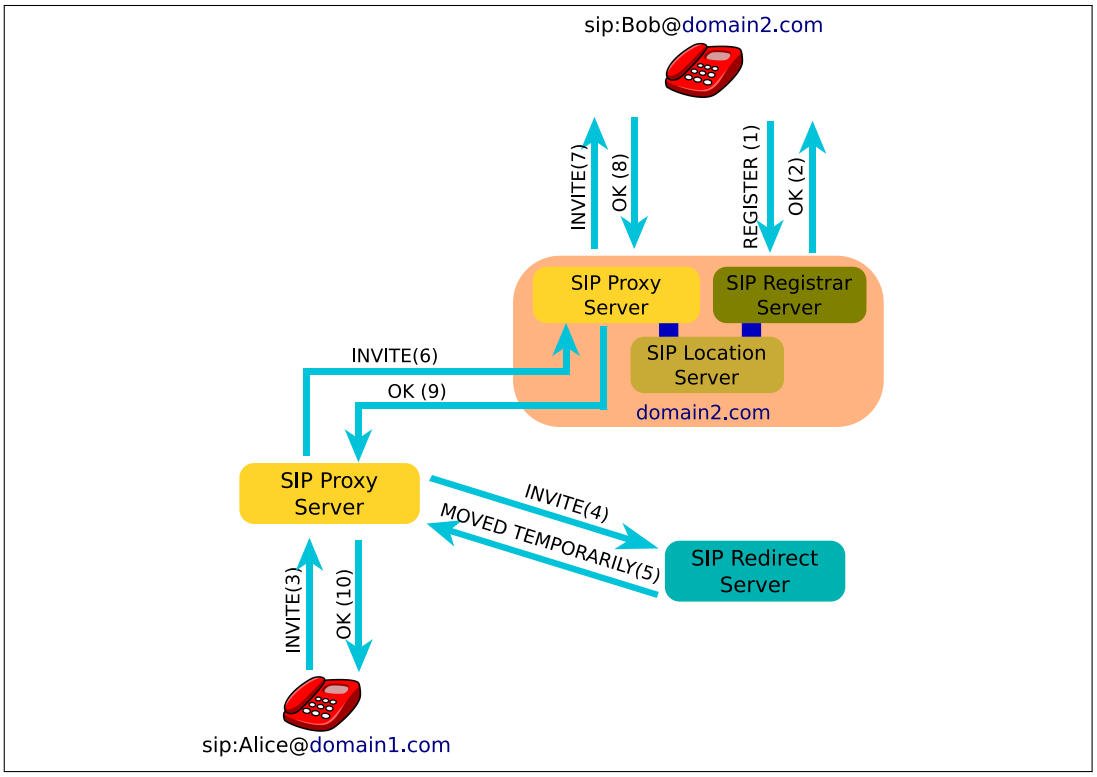


Figure 1.2: SIP architectural components

users from the domain. However, UAs do no directly interact with the location server but indirectly by means of proxies, redirect or registrar servers. Basically, it is used by SIP servers to allow application level mobility.

- **SIP Servers:** these components provide a vast range of extra functionalities to facilitate the establishment of a session between two SIP UAs. According to their functionality, SIP servers can be logically subclassified as follows:
  - **SIP Proxy Server:** used to forward requests on behalf of other SIP entities. It can not initiate a request by itself, but can offer additional services like for instance security, authentication and authorization.
  - **SIP Registrar Server:** receives the request from a UA which wants to register in its SIP domain. Then, the server updates the location server with the UA's information, the user name, location address, preferences, etc. for future use.
  - **SIP Redirect Server:** used to indicate the location where the initial request has to be forwarded. It is mainly useful for mobility purposes. The difference with respect to the proxy is that the Redirect Server tells the entity the contact address (of the UA) rather than forwarding requests itself. The redirect server is also able to retrieve multiple locations to allow the proxy to fork the call.

1.1.2 Functionality

SIP has been designed to establish, modify and teardown sessions. Only session signaling is considered by SIP, however it can be used in combination with other protocols to build a complete multimedia session. SIP does not rely on any specific transport layer. It neither imposes the

media protocols, for instance, the Session Description Protocol [75] (SDP) is used for arranging the media session. Usually, RTP is used to transport the media session. According to RFC 3261 [120], SIP supports five distinct features:

User location: identification of the end system to be used for the session. The protocol transparently deals with participants which can move from one location to another, or they can be reachable at several addresses.

User availability: requires parties approval to engage them in the session.

User capabilities: identification of which features are supported by each party.

Session setup: determination of parameters used by each of the parties.

Session management: modification of ongoing parameters in the session, tearing down the communication, transfer and invoking additional services.

SIP is a Request/Response protocol. In a normal session many requests can be generated where the request itself is identified by a *Request Method* and the possible responses (provisional or final, see below) by the *Response Codes*. Table 1.1 illustrates the most common request methods defined in SIP, new methods may be defined to extend the protocol functionality.

REGISTER	register the end point into a SIP domain. Thus the registered party is is reachable from the Registrar server
OPTIONS	used to query the capabilities supported/aliveness of each party
INVITE	used to initiate a session between parties or to modify an ongoing session
UPDATE	modifies the state of the session before it has been accepted. Thus, it only exists between an INVITE and its corresponding final response
CANCEL	cancels a pending call that has not yet been accepted
BYE	finishes the current session
REFER	used to transfer a call
MESSAGE	transports an instant message
INFO	sends information in a session already established using INVITE. For instance, it can be used to send dial tones
SUBSCRIBE	used to subscribe to event notifications
NOTIFY	transports the notification event
PUBLISH	post an event into the server responsible, interpret and distribute this event
ACK	used to acknowledge final responses
PRACK	used to acknowledge provisional responses

Table 1.1: SIP request methods

Responses to a request can be classified in two types: provisional and final. Provisional responses provide information about the current status of the request but are not sent reliably (i.e. no acknowledge response is needed). Final responses deliver information about the result of the request and they are sent reliably. The response information is coded in six different categories as described by table 1.2.

1xx	Informational responses which are used to acknowledge that the request is being processed, that the user has been notified (e.g. the device is ringing), etc.
2xx	Successful responses
3xx	Redirection responses used when the services have been moved (either permanently or temporarily) or any other mean to inform that the messages have to go by another route and/or another destination
4xx	Client failure responses used to report errors in the message format or content, or to disallow such request in the current state (e.g. requiring authentication)
5xx	Server failure responses to describe unsupported methods, internal errors, etc.
6xx	Global failure responses

Table 1.2: SIP response codes

1.1.3 Message Format

SIP has been designed on the same basis as HTTP where each message is human readable. Depending on the nature of the message (Request or Response), it is composed of: a Request Line or a Status Line, several message headers and a message body as described in the following ABNF:

```
SIP-message     =  Request / Response

Request         =  Request-Line
                          *( message-header )
                          CRLF
                          [ message-body ]

Response        =  Status-Line
                          *( message-header )
                          CRLF
                          [ message-body ]
```

The Request-Line indicates the request type (as described in table 1.1), the entity for which the request is generated and the SIP version being used. The Status lines is composed of the response code (as in table 1.2) and the SIP version being used. In fact, many of the Status codes have been reused from to the ones defined in the HTTP protocol.

The messages headers are used to provide the information about the current session, the local and destination parties and possible hops in the transport, routes to follow, etc. A valid SIP request must contain, at least, the *Via, To, From, Call-ID, CSeq* and *Max-Forwards* headers (described in table 1.3).

Finally, the SIP message can carry other types of information encoded in the message body (Figure 1.3 lines 19 to 31). This information is, for instance, used to agree on the media codecs of the session using SDP. However, other contents can be carried for different purposes.

In Figure 1.3 a complete SIP message is illustrated. The message is a session initiation request; started from Alice and directed to Bob, in which Alice already specifies which codecs her actual phone is able to use (i.e. the SDP content).

It is worth noting that the *To* header represents the original destination of the request, while the *Request-URI* can be modified due to call forwarding or other proxy operations.

Via	This header contains the SIP version and the transport protocol been used, together with the address and an identification tag (known as branch) of the server that forwarded this request (as shown in Figure 1.3 line 2). One additional Via headers is added for each SIP component who forwarded the message
From	This header contains a display name, a SIP address from the entity that generates the request together with an identification tag (as shown in Figure 1.3 line 3)
To	The same as the From header but concerning the entity to which the message is addressed (as shown in Figure 1.3 line 4)
Call-ID	This header contains information that uniquely identifies a session (SIP dialog) as explained in section 1.1.4 (as shown in Figure 1.3 line 5)
CSeq	The CSeq header contains an integer followed by a request method which links the message to a specific SIP Transaction (as explained in section 1.1.4). Shown in Figure 1.3 line 6.
Max-Forwards	This header specifies how many SIP hops the message can pass before being discarded. Shown in Figure 1.3 line 7.

Table 1.3: SIP message header fields

1.1.4 Hierarchy

SIP messages can be classified in a hierarchical way according to Dialogs and Transactions.

SIP Dialogs: the SIP RFC defines the dialog as to represent a peer-to-peer relationship between two user agents that persists for some time. A dialog is identified by the Call-ID value, local tag and remote tag of the message. The local and remote tag are respectively identifiable on the From and To header.

SIP Transactions: each transaction is defined to be a single request and any response to that request, which includes zero or more provisional responses and at least one final response. In the case where the request transaction is an INVITE, the transaction will also include an ACK message only if the final response is not a 2xx (successful response). If the final response is not a 2xx then the ACK is not considered to be part of the transaction.

One SIP dialog may contain several SIP transactions. Each transaction may be related to a previous one given the protocol its stateful context (e.g. hanging up only after the call has been established). A transaction is identified inside a dialog by the CSeq and the parameters of Via headers (specifically the branch tag).

Figure 1.4 illustrates a simple dialog between two entities in which the caller (Alice) initiates a session with the callee (Bob). They establish a Media Session and finally the caller hangs up the phone.

1.1.5 Authentication

Each request message of SIP, except CANCEL and ACK, can be challenged for authentication. Thus, the VoIP services can be protected against threats and attacks like impersonation, session teardown, fraud and others [116, 133]. Authentication in SIP has also been leveraged on the design of HTTP authentication. It is based on a challenge-response scenario.

```
INVITE sip:Bob@domain.com SIP/2.0                                    SIP
Via: SIP/2.0/UDP pc1.domain.com;branch=z9hG4bK34
From: <sip:Alice@domain.com>;tag=as07b23bqd
To: <sip:Bob@domain.com>
Call-ID: 000b46d9-cb860180
CSeq: 100 INVITE
Max-Forwards: 70
User-Agent: IP_Phone/8.0
Contact: <sip:Alice@domain.com;transport=udp>
Expires: 180
Accept: application/sdp
Allow: ACK,BYE,CANCEL,INVITE,NOTIFY,OPTIONS,REFER,REGISTER,UPDATE
Remote-Party-ID: "Alice" <sip:Alice@domain.com>;party=calling;
Supported: replaces,join,norefersub
Content-Length: 276
Content-Type: application/sdp
Content-Disposition: session;handling=optional

v=0                                                                  SDP
o=SIPUA 12941 0 IN IP4 192.168.1.20
s=SIP Call
t=0 0
m=audio 32250 RTP/AVP 0 8 18 101
c=IN IP4 192.168.1.20
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
```

Figure 1.3: SIP message

Authentication challenges are computed using pieces of information extracted from the authenticate message plus the username and a shared secret. In the simplest case, the authentication *response* is computed as follows:

```
A1       = username ":" realm ":" passwd
A2       = Method ":" Digest-URI
response = MD5(MD5(A1) ":" nonce ":" MD5(A2))
```

The computed authentication response will be rejected if either the username, password or any of the entries do not match the ones obtained from the challenged message. It is worth noting, that the nonces should be one-time used just to avoiding an attacker to reuse the challenged response for making unappropriated calls.

Figure 1.5 illustrates the challenge-response process between one UA and a SIP proxy. In the process the caller (Alice) tries to initiate a session with Bob via its own proxy. The proxy will ask Alice to authenticate to a generated challenge given in the **401 Authentication required** request. In the following, the caller must complete the challenge and generate a new **INVITE** carrying the response in order to continue with the session establishment.

It is also worth noting that man-in-the middle attacks are still possible if the channel is not encrypted.

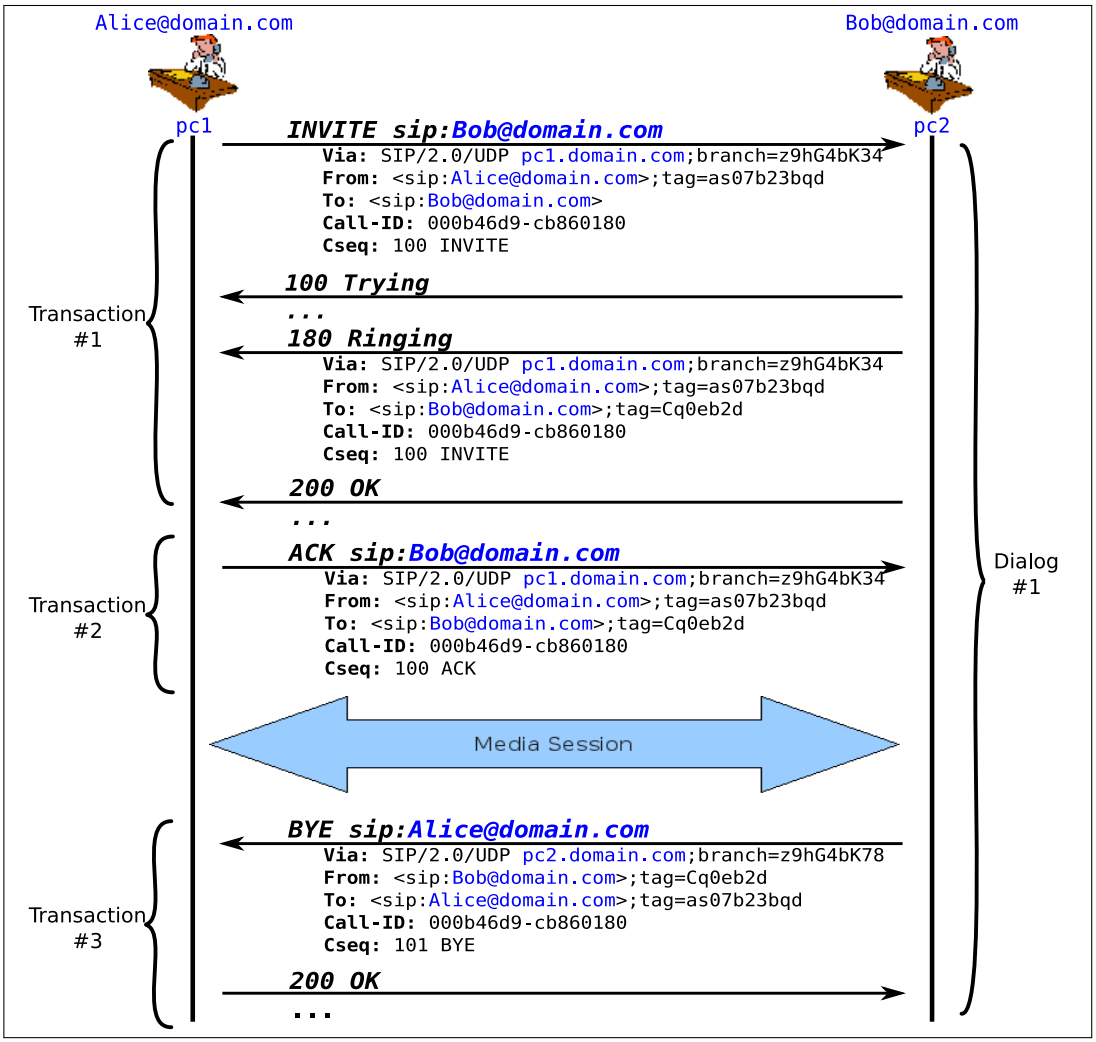


Figure 1.4: SIP dialog and transactions

1.2 VoIP Threats

"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts"

Gene Spafford [3]

All sort of deployed equipments are exposed to different security risks. The degree of exposure is associated to how external factors can stimulate a component (e.g. the people having direct/indirect access to the component) and how the system treats those factors and what effect they trigger on it. Thus, a threat can be classified as a menace existing in the system, while a vulnerability is a security breach that can be used to compromise it. In practice, threats and vulnerabilities are related to several factors, among them we can identify design and implementation errors, misconfigured software, experimental features deployed, etc.

With all the new software components delivered (most of them do not fulfill all security standards) and with the Internet, the number of disclosure of threats and vulnerabilities has notably

[3]Source: Scientific American, 1989, pp 110, Computer Recreations: Of Worms, Viruses and Core War by A. K. Dewdney

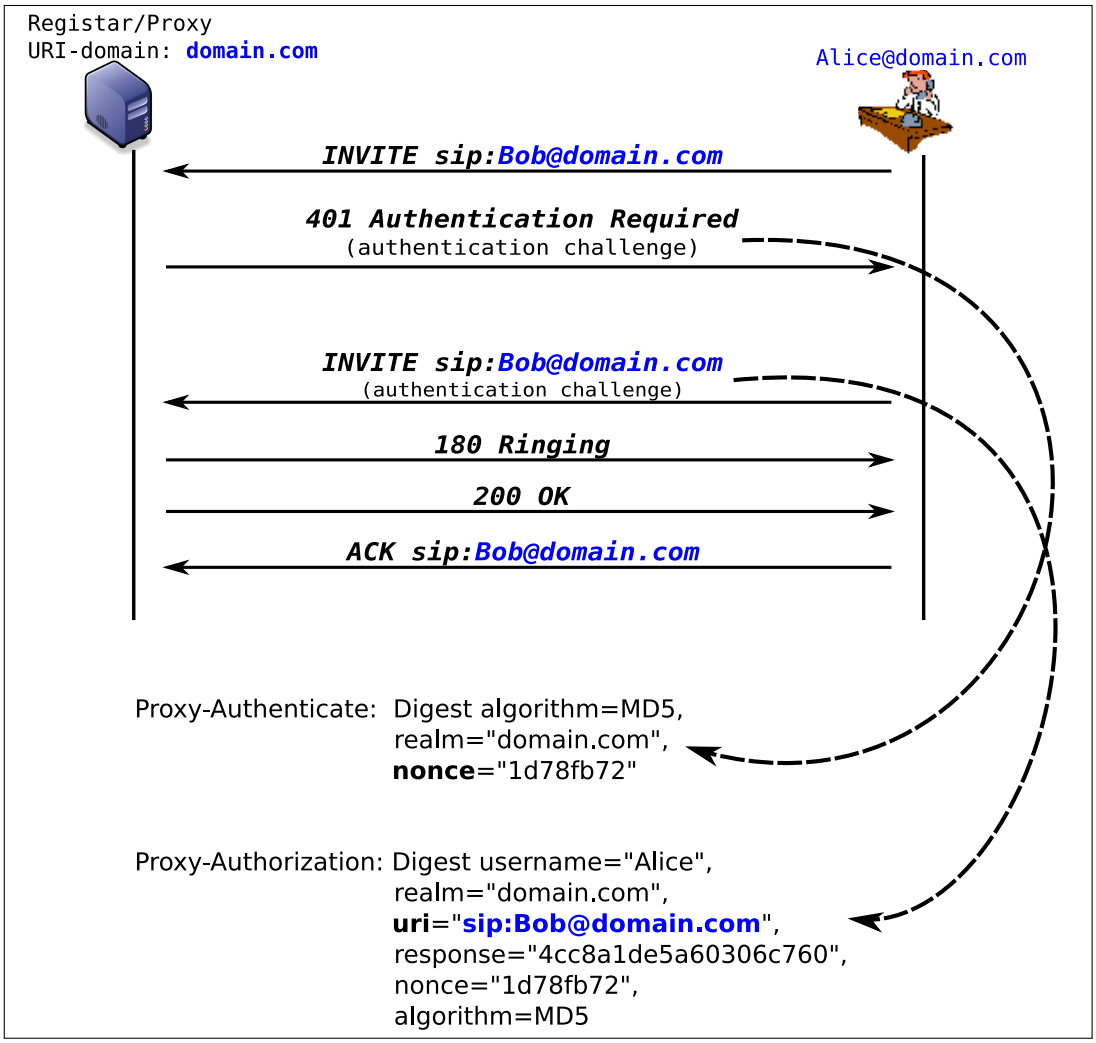


Figure 1.5: SIP authentication

increased over the years, as illustrated by figure 1.2 extracted from the US-CERT database[4].

All those vulnerabilities shown in the figure are only the ones publicly disclosed, and they can be classified in six categories following the STRIDE threat model defined by Microsoft [78]:

Spoofing identity when someone else authentication is illegally used.

Tampering with data when alteration of data is illegally committed.

Repudiation when traces of illegal operations are properly hidden or erased.

Information disclosure when data is exposed to individuals which are not granted.

Denial of Service when an user is not able to use a service which is provided to him/her.

Elevation of privilege when somebody can gain privileged access to a system where he/she is not supposed to.

[4] `http://nvd.nist.gov` last checked December 2008

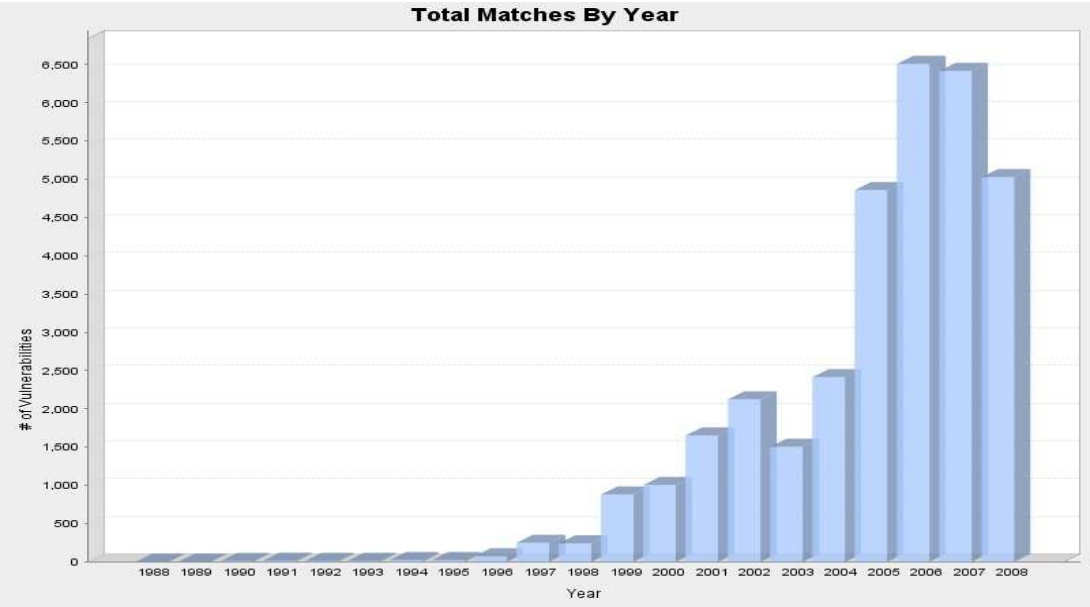


Figure 1.6: Software flaws (CVE) from 1988 until 2008

As the community of VoIP started to increase, the security awareness for such systems became of importance. The Voice over IP Security Alliance[5] (VOIPSA) was therefore created with the purpose of promoting, educating and providing methodologies and tools for people using VoIP services. VoIPSA has referenced several potential security threats for VoIP deployments [58]. They are described below.

Social threats become an important concern in security and privacy. Different threats are included in this class like misrepresentation which includes false information delivered expressly with the intent to mislead. For instance, misrepresentation may range from intentionally presenting a false identity as if it were true (e.g. caller ID, voice, contact information), showing information from somebody else as its own with the objective to bypass authentication mechanisms (e.g. passwords, usernames, organization) or to modify the content of the information, to mislead the origin of the call (e.g. scams, phishing). Theft of services is another threat related to any activity which unlawfully tries to gain revenues from someone else's services (e.g. billing records alteration, service abuse). Finally, contacting one entity without its prior consent, if required, or bypassing such consent is also part of this category.

Eavesdropping is a method employed by an attacker in order to obtain, monitor or reconstruct any kind of information exchanged between any other entities. It may include analysis of call patterns, traffic capture or any type of message reconstruction (e.g. voice, fax, video, text).

Interception and modification attacks require to both observe all or part of the traffic between two or more entities, and be able to modify such conversation or signalling. Instances of attacks directed to the media data can be related to fax & conversation alteration, impersonation and hijacking or any means to alter partial or total information in the content of the message. In terms of the signalling, different attacks from this class can be observed

[5] `http://voipsa.org` last checked December 2008

like Call Black Holing; when information is hidden or dropped, intending to prevent termination of the call under normal circumstances. Call Rerouting is another attack in which the flow of the call is directed over unauthorized routes. Therefore, unauthorized nodes can be added in the flow or even authorized ones will be removed from it. As a consequence, this attack can, among others, degrade the conversation quality or hijack the session.

Service abuse is concerned with any improper use of the VoIP services. For instance, call conference abuse can be used to hide the identity and therefore commit frauds. Other types of abuses such as bill alteration or bypassing as well as modification in the signaling process are included in this threat.

Physical access is not only limited to breaking into the company facilities, it may be related to degradation of the services when access to the physical network is gained. For instance, obtaining access to the systems and equipments as well as to conduct man in the middle (MiM) attacks, configuration changes and loss of power.

Interruption of services covers a vast range of threats from loss of power, resources exhaustion to denial of services. These attacks can be performed by different means like flooding the services (either with malformed messages or not), looping (in which messages are constantly forwarded back and forth), sending malformed requests (which may trigger implementation vulnerabilities), faking responses (which can maliciously teardown a session), degrading the Quality of Service (QoS), etc.

1.3 Current Best Practices

VoIP networks are subject to more attacks than the PSTN. VoIP also introduces new threats in IP networks if they are not well deployed. However, careful planning and proper management can mitigate such risks and allow to benefit from VoIP services. Several best-practices recommendations have been written to minimize such risks [6, 87, 107, 114, 116, 133]. Such practices require specific design in the network and engineering work to achieve certain levels of security. Such measures describe actions like applying the current patch to known flaws or vulnerabilities, use of Anti-Virus and Intrusion Detection Systems (IDS) and update them regularly, use of firewalls and Application-Layer Gateways between trusted and untrusted zones, implement authentication, encryption and/or Virtual private network (VPN) on critical segments and isolation of VoIP services via Virtual LAN (VLAN).

1.3.1 Deployment & Maintenance Support

Nowadays, as vulnerability disclosures are accessible to everybody, administrators must be informed and prepared for any known/unknown threat that could exists in their network. Then, several considerations have to be taken before and after the deployment of a VoIP network (it may apply to any media network as well).

Physical security: access to the main equipments (including router, server, databases, etc.) must be restricted and isolated.

Physical robustness: alternative routes, backup equipments/configurations must be maintained at all time.

Management security: strong authentication/encrypting algorithms must be used from both trusted or untrusted networks.

Equipment update: keeping up to date security patches or remediation techniques to defenses against subsequent attacks.

Log analysis: traffic must be analyzed for the purposes of finding abnormal behaviour in the network. This tasks must be assisted with the use of IDS and monitoring techniques.

Assessment: systems must be periodically tested using attacking techniques to identify weaknesses of the network and possibly find mitigation techniques.

1.3.2 Protocol Protection

Since signalling has been shown to be susceptible to numerous attacks, SIP recommends different alternatives for securing the integrity and content of transport and network layers. There are two methodologies that can be used for this goal: hop-by-hop and end-to-end. Hop-by-hop encryption assumes all the intermediaries are trusted sources and support the encryption mechanism. The encryption properties are maintained between two consecutive entities and then, for each hop a different encryption instance must be created. In case one of the hops does not, the traffic will be sent in clear text losing its confidentiality. End-to-end encryption is simpler, but is mostly inadequate for SIP when intermediary elements need to observe/modify certain values of the message (e.g. for routing the message correctly). Below the recommended encryption techniques for SIP are described:

SIP over TLS: the Transport Layer Security (TLS) [53] protocol has been conceived to provide data integrity and confidentiality for TCP traffic between two consecutive hops. It is based on the use of certificates provided by a trusted third party (Certificate Authority (CA)) or on the use of pre-shared keys. In SIP this mechanism is supported by the use of SIPS-URIs (Secure SIP or SIP over TLS), slightly modifying the syntax of source, intermediary and destinations addresses. This makes the implementation almost transparent to the SIP application.

Although its usage has a low performance impact on SIP, there are a few limitations that must be considered. It is a hop-by-hop mechanism and mutual authentication may not be scalable [128] since it requires the deployment of a full-fledged Public Key Infrastructure (PKI) [15]. Finally, it has been designed to be used with a reliable transport protocol, therefore requiring all applications to use TCP or Stream Control Transmission Protocol (SCTP) [130].

SIP over DTLS: the Datagram Transport Layer Security (DTLS) [118] protocol provides equivalent protection mechanism than TLS using an unreliable transport layer, like UDP. Recently, work has taken place in order to define a standard for the use of this protocol with SIP [82]. It benefits and suffers from the same properties than TLS. However, there is no known implementation yet.

IPSec: IPSec (IP Security) [85] allows to create a secure tunnel between end points, thus providing mutual authentication, encryption, anti-replay and data integrity. It can be used end-to-end or hop-by-hop and it can be used without knowledge of the application it operates for the network layer. It had been demonstrated in a study done on behalf of the NIST [32] that establishing a SIP call using IPSec tunnels incorporate a delay of $\approx$3 seconds. In a scenario where Alice calls Bob via two intermediaries proxies (assuming each UA are located behind their own proxy) the delay can take up to $\approx$20 seconds. However, industry standards require that this time should not exceed 250ms.

S/MIME: Secure/Multipurpose Internet Mail Extensions [115] (S/MIME) message body encryption can be used in an end-to-end fashion. This method relies on a Certificate Authority (CA) to proceed with the encryption and can provide authentication, integrity, encryption and non-repudiation of origin. It is independent of the transport layer and it may allow to encrypt portion of content of the SIP message (except for the Via headers which must always be readable). It requires a PKI deployment making it difficult to scale.

Securing the VoIP traffic is critical and it must not be limited to just the signaling information. Thus, different techniques must be addressed to preserve the integrity and confidentiality of the media data.

SRTP: the Secure Real Time Protocol (SRTP) [123] provides authentication, confidentiality, integrity and replay protection of the media data. SRTP defines a mechanism for which session keys are derived, however it does not define the key management. Multimedia Internet KEYing (MIKEY) [81] addresses a real-time key management scenario for which it became the recommended solution.

ZRTP: is defined as a key management scenario for SRTP. It uses the Diffie-Hellman key exchange protocol during call setup (the exchange is conceived in-band). It does not rely on a PKI neither on CAs; the keys are exchanged in the session and displayed on the telephones screen as a short authentication string. Then, the users must read the string out aloud; if they match, then no man in the middle attack has been performed.

1.3.3 Network Design

All recommendations propose the segregation between VoIP and data networks as well as dedicated access control mechanisms. However, deployment of such practices may require extra considerations in order not to disrupt specific services. For instance, when segregating the traffic there may be some links between both networks that still need to be maintained (e.g. the voice mail or the call records). This establishes a difficult challenge for VoIP administrators to maintain their networks secure. Intermediary entities, such as firewalls, may need to look in the content of body messages to do certain actions like opening the appropriate ports. Since media ports are selected randomly, static rules are not useful. Therefore, Application-Layer Gateways (ALGs) may be deployed to deal with such issues. They can dynamically open and close ports in the firewall, preventing scanners detecting the topology and services running inside the network. Also, ALGs can be used to filter abnormal or malformed traffic.

One possible approach to isolate the VoIP network, is for instance, to use VLANs. Thus VoIP traffic is prioritized and can be protected from data network attacks. However, as VLANs do not support user authentication, information can jump from one network segment to the other. Another approach is the use of VPNs for sensitive information, where a call can be encrypted/decrypted selectively. This may protect against attacks derived from spoofing or sniffing information. However, as a consequence the Quality of Service (QoS), can be affected by the cryptographic processing. The physical design of the network supported for the VoIP services also plays an important role, Figure 1.7 exhibits a possible architecture.

Table 1.3.3 illustrates the mentioned best practices approaches and risk.

1.4 Summary

Voice over IP services are becoming widely used mostly due to the vast range of features that they provide and their low cost. However, such a vital service can not afford to suffer a “system down”.

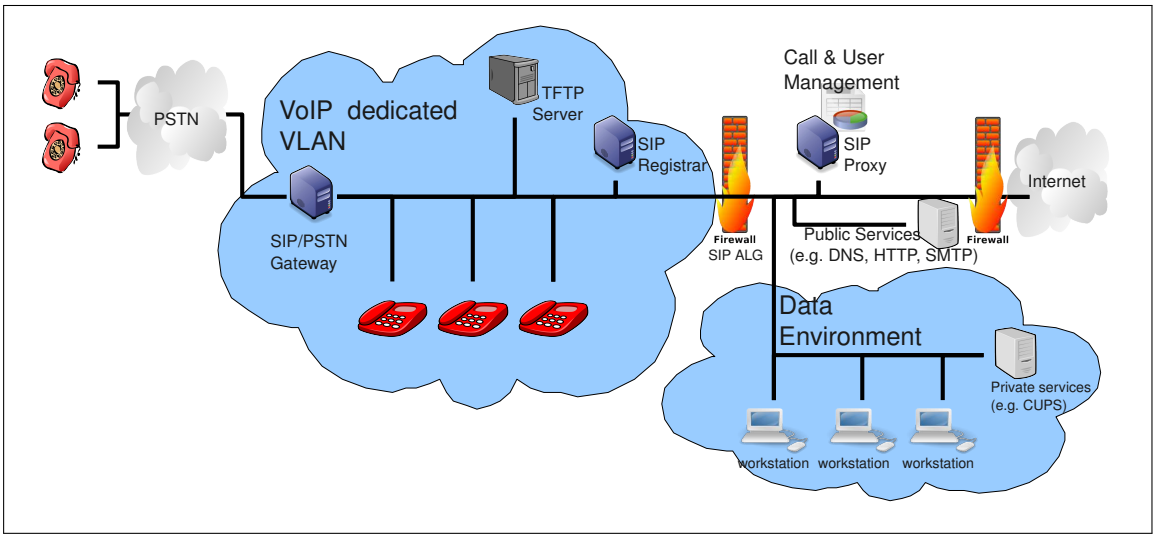


Figure 1.7: Network services security architecture

Threats	**Mitigation**
Social Threats	Personal training
	IDS, ALG
	Black & White lists
Eavesdropping	Signalling encryption (TLS, DTLS, IPsec, S/MIME)
	Media encryption (SRTP, ZRTP)
	Data Segregation (VLAN, VPN)
	(Pro/Post)-Active testing
Interception/Modification	Data Integrity (TLS, DTLS, IPsec, S/MIME, SRTP, ZRTP)
	Data Segregation (VLAN, VPN)
Service Abuse	IDS
	Strong authentication
	Assessment
Physical Access	Robustness and access control
Interruption of Services	Maintain up to date all the network nodes
	IDSs, ALGs
	Assessment
	(Pro/Post)-Active testing

Table 1.4: Best-Practice approaches and risk

Imaging trying to call an ambulance, police or any emergency entity while your phone/provider says *"Sorry for the inconvenience, we are rebooting"*. The availability expected from the already deployed PSTN is 99.999%, currently VoIP services are one order of magnitude lower [5]. Indeed, the problem of emergency calls is that IP networks make the resolution of the geographical location of the call difficult. This complicates the routing to the nearest call center [106] (imagine using a VPN connection).

VoIP has demonstrated to be practical and it gained an incredible popularity. Nowadays, security flaws are emerging due to the lack of testing from quick implementations, inexperienced administrators and bad deployments. Even, if the previous statement is quite pessimistic, it is

just a matter of time until VoIP replaces the PSTN. During that period, we are encouraged to search for all type of vulnerabilities, defects and possible inconsistencies in order to make these services more robust.

This thesis phased different approaches, first it aims to organize assessment techniques in order to test and monitor deployed networks. Second, it aims the remotely identify equipments based on particular properties presented in their messages. Third, it aims to the search of software vulnerabilities before and after the devices are deployed using active testing algorithms.

Chapter 2

Security Assessment

Network security assessment provides the necessary knowledge to the administrators in order to estimate the degree of security that their networks reach. Different concepts of assessment range from inventory management to almost full fledged penetration tests [102]. The main objective of an assessment is analyze the whole system under evaluation in order to find all the potential vulnerabilities. Instead, a penetration test will be limited to find enough number of holes to exploit such vulnerabilities. In terms of methodologies used, there is a thin borderline between penetration testing and assessment. Even further, an assessment test is not supposed to exploit vulnerabilities with a proof of concept code, but in practice in order to avoid false positives, this is sometime crossed.

The best approach to avoid intruders to get into the network is to have the latest software updates, restrict already known vulnerable services and the most important is to periodically perform comprehensive assessment tests. For these tests, network devices and hosted services must be automatically discovered by the assessment team and followed by the tests that are planned to be be performed.

For illustration purposes, consider that a service by itself can provide a high level of security, but when it is dependent on/or in relationship with others, a security breach might occur due to their interaction. Information about the type of entities, the services on which they are dependent as well as the hosting device is a starting point to discover flaws in the target system.

For instance, a simple SIP hard phone may depend on a DHCP service to obtain its IP address, on the DNS service to resolve IP addresses, on a TFTP service to retrieve its configuration and firmware, and on a SIP Proxy/Registrar to setup and receive calls. If one of those services is impersonated or compromised, the overall functionality of the phone may become insecure.

To network administrators, many pieces of information regarding the topology and the configuration of devices in the network are known. However, maintaining a static description of services running on each network node is quite unrealistic since users are prone to install new applications. This information can be useful to make the assessment but it must be complemented with network discovery tools which infer the presence of devices through advanced TCP/IP and application level fingerprinting tools.

As a reminder, security assessment is less invasive than a real attack even though it has to be carefully planned in order not to disrupt services and compromise the whole network.

Both a methodology as well as supporting tools for security assessment are becoming of special interest[6]. Some recent books [114,116,133] as well as several technical reports [6,87,107] describe the operation, best practices and recommendations for securing VoIP networks and services without, however, providing guidelines on how to test them.

[6] `http://voipsa.org/Resources/tools.php` last checked on December 2008

Figure 2.1 shows a flow in which a network assessment campaign takes place in the network and the following sections explain each component in detail.

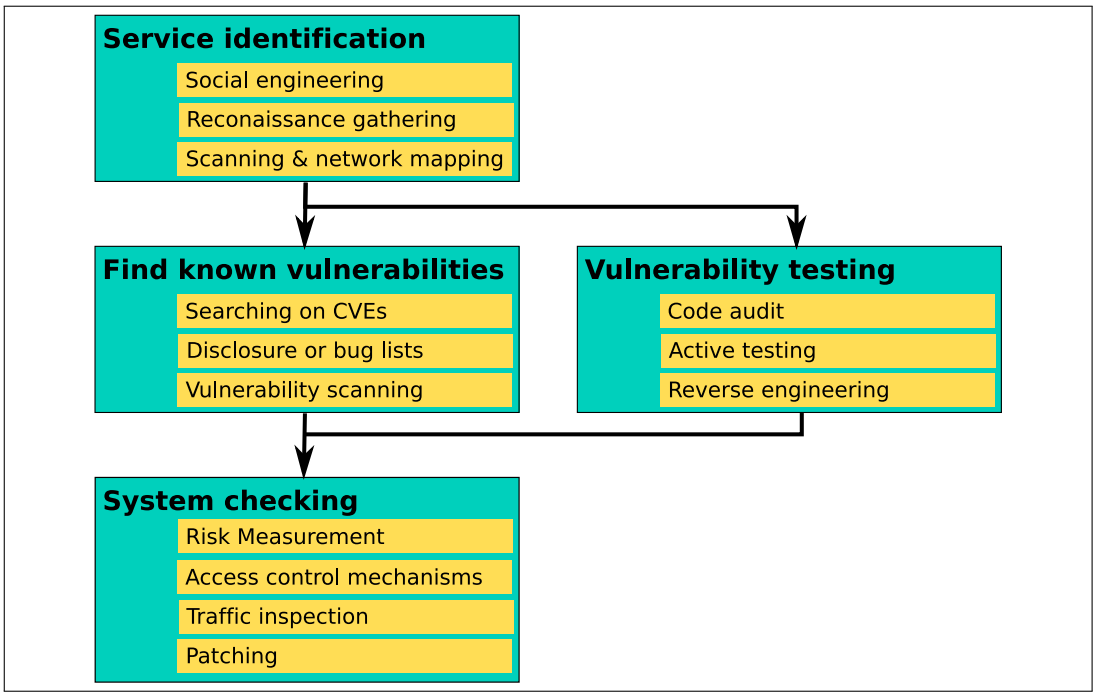


Figure 2.1: Network assessment

2.1 Service Identification

The goal of this phase is to obtain host OS, topology and services information from the network environment. Once the environment has been investigated, the flaws of the system can be identified and be abused in later stages. Different types of identification methodologies are considered for this phase and they are described below.

2.1.1 Social Engineering

Social engineering techniques abuse the tendency of human beings to trust. Its goal is to obtain personal benefits on behalf of the target people. There are different techniques described in the literature (for instance [71, 111]), but they lie outside the scope of this thesis. In brief, the used techniques include intimidation, impersonation, blackmail, deception, flattery, befriending, authority, pressure, vanity and/or sympathy.

2.1.2 Reconnaissance Gathering

Reconnaissance techniques try to obtain as much information as possible from the company, its employers and its IT infrastructure. For instance, professional & personal web pages from employers can provide internal or proprietary data. Also, several public registries (e.g. European IP address allocation [7], US Army[8]) or utilities (e.g. whois, traceroute, nslookup, dig, host) can

[7] `http://www.ripe.net` last checked on December 2008
[8] `http://whois.nic.mil` last checked on December 2008

be used for collecting such data. These tools and sites may allow the attacker to discover information such as administrative contacts, network domains, allocated IP addresses, name servers, mail servers, etc.

2.1.3 Scanning and Network Mapping

This step aims at inspecting the network architecture without any knowledge of it - as if it was an attacker who is inspecting the network. Its main objective is to discover the network topology (i.e. firewalls, routers, VLANs), active systems (i.e. application servers, IDS, configuration servers) and OS information (i.e. Linux/Windows/Cisco IOS, open/closed ports).

Networks scanners lead the task of recognizing the network topology. Basically, they search for listening ports at random or specific addresses. The process consists in sending traffic and waiting for a timeout or a response. If there is a timeout, then the host does not exist, the message was filtered or it has been dropped. If the host replies, it means that the port is open, closed or not listening. Among the most popular scanning tools [9], we find Nmap, Superscan, Unicornscan.

The process of identifying an entity or service from properties of the generated messages, is called Network Fingerprinting. Fingerprinting systems can be classified in either active or passive. A passive fingerprinting tool only listens to the traffic on the network and processes it with the goal of recognizing the actual state and the device that generated the message. In contrast, through the injection of crafted/normal packets, the active fingerprinting method tries many techniques to discover information from the host.

In the following, we make a brief description of two of the most popular tools for such phase.

Nmap Active Scanner

One of the most recognized tool for remote OS detection and port scanning is Nmap[10]. Its discovery procedures are based on exact TCP/IP network stack fingerprinting. This is done by sending multiple packets to the target machine and examining specific fields in the answered TCP packets. Such packets belong to sample tests to discover supported options and vendor specific stack behaviour. Once those tests are answered, the received messages are analyzed and a fingerprint entry is created.

A signature entry is composed of many categories and is made of a list of attribute and value pairs. A value as well as a whole test can be missed depending on many factors as, for example, that the test is not supported by the system or if no reply was received, etc. The result of the test includes categories like SEQ (sequence analysis of the probe packets), OPS (TCP option received from the probes), WIN (windows sizes of the probes), etc.

Once the fingerprint entry is created it is compared with the OS or service fingerprint signatures in the Nmap database to identify the device.

This type of discovery is very disrupting since most of the sent packets violate the specifications of the given network protocols. The information obtained by such a tool includes the running operating system, the open/closed/filtered ports as well as the services hosted on them, the version of the software, etc...

[9] `http://sectools.org/` last checked on December 2008
[10] `http://www.insecure.org/nmap/` last checked on December 2008

P0f Passive OS Fingerprinting

P0f[11] is a passive OS fingerprinting detection tool. As a passive fingerprinting tool, it listens to the packets transiting on the network. It analyzes the different behaviours and peculiar protocol level specifics presented by the target implementation. It uses fingerprinting techniques over the incoming and outgoing packets of a connection as well as the information observed from refused and established ones. Through this, it will identify the software or/and version running in the target devices. P0F provides a mechanism to add new fingerprint signatures which depend on the combination of different values of the packet. The signature entry for a TCP/IP packet looks like:

```
wwww:ttt:D:ss:OOO...:QQ:OS:Details
```
(2.1)

as depicted in Table 2.1.

wwww	window size. It can be *, %nnn, "Snn" (multiple of MSS) or "Tnn" (multiple of MTU) are allowed.
ttt	initial TTL
D	don't fragment bit (0 - not set, 1 - set)
ss	overall SYN packet size
OOO	options in the order they should appear in the packet.
QQ	list of oddities or bugs of this particular stack.
OS	OS type (Linux, Solaris or Windows)
details	OS description (2.0.27 on x86, etc)

Table 2.1: TCP/IP signature entry for p0f

Every packet captured by P0F will be analyzed and compared with the list of signature entries and filtered to match the corresponding OS of the device.

Since this is a passive tool, no packet is injected into the network and thus the tool can hardly be detected. As a consequence it depends on the packets sent by the hosts, which makes it often less accurate than active tools.

However, one important remark on P0F is that it can detect fingerprints through a firewalled connection, where active scanners will fail.

2.2 Finding Known Vulnerabilities

This phase consists in identifying the known vulnerabilities or security breaches that exist in the network. Different techniques are mentioned below.

2.2.1 Vulnerability Searching

Nowadays, Internet contains several places were information about security vulnerabilities can be found. The flaws are not always disclosed with a patch, mitigation or anything to avoid their exploitation. Among the typical sites we find the Common Vulnerabilities and Exposures[12] (CVE),

[11] `http://lcamtuf.coredump.cx/p0f.shtml` last checked on December 2008
[12] `http://cve.mitre.org/` last checked on December 2008

disclosure lists[13], vulnerabilities auctions [14] and bugs & patches forums from the products.

2.2.2 Vulnerability Scanning

Vulnerability scanners were introduce in 1987 by R. Baldwin [21] with the U-Kuang system, where its goal was to identify operational security problems running in a host. In 1990 D. Farmer and G. Spafford [59] described the Computer Oracle Password and Security System (COPS). COPS is a composition of several tools (including U-Kuang) which objective was to provide more instrumentation to administrator to assess a host. Further work of D. Farmer and W. Venema conclude with the Security Administrator Tool for Analyzing Networks[15] (SATAN). SATAN was the approach which allowed administrators to analyze their own network from an outsider perspective. The methodology consists in first checking which are the active hosts in the network; including open ports and hosted services as it is done by the network scanner tools. Secondly, it tries to identify running versions and patch level of softwares. Finally, when the reconnaissance is done, it may - or may not - test if the vulnerability is effective by launching the exploit. Note that if a tool of this type is launched by the administrator, it has to be used with special care since it can disrupt the normal functioning of the network.

Since SATAN, several tools have emerged trying to provide more features and more complete set of attacks[16]. Among the listed tools, Nessus[17] is one of the most popular vulnerability scanner and it is briefly described below.

Nessus Vulnerability Scanner

Nessus is a vulnerability scanner designed as a plug-in architecture. It works by first identifying the network components, open ports and hosted services (including their running version). It can use its own port scanner or rely on the results of external ones (for instance Nmap). Afterwards, it can trigger several attack scenarios like: remote control, missing patches, misconfiguration, default passwords & dictionary attacks, denial of service, etc. Nessus includes its own attack language (NASL: Nessus Attack Scripting Language) which allows users to write their own exploits for specific contexts.

2.3 Vulnerabilities Testing

This phase looks for threats, vulnerabilities or bugs which are still unknown to the public. Different entities may be interested to perform this procedure. For instance, a company may want to improve the quality of its products over time; a consumer concerned by its own network security may run such tests before deciding which equipment to buy. Indeed, it may periodically test its equipments after having deployed them . And finally, security companies or researchers may test equipment to publicly validate their techniques.

Different approaches (described below) may be used in this phase depending on budget, code availability or personal preference.

[13] `http://seclists.org/fulldisclosure/` last checked on December 2008

[14] `http://www.wslabi.com` last checked on December 2008

[15] `http://www.porcupine.org/satan/` last checked on December 2008

[16] `http://www.securityinnovation.com/security-report/October/vulnScanners15.htm` last checked on December 2008

[17] `http://www.nessus.org/` last checked on December 2008

Code Audit

Code auditing is used to check structural errors in the source code of an application. It usually searches for common vulnerabilities that may exist in the implementation since some programming structures may require careful attention. Typically functions like strcpy, printf, scanf, etc. or casting could lead to overflow vulnerabilities [20, 111]. Incorrect input validation can lead to SQL Injection [95] or Cross Site Scripting attacks [125]. Code auditing may find several flaws in the code but not necessarily all. Indeed, it requires a high amount of work and it may report many false positive alarms.

Reverse Engineering

Reverse engineering [40] is the process of inferring the software structure out of the closed-source binary application. It observes in details the functional behavior of each component in order to deduce how it is implemented. This technique can be used to understand the functionality of the application, extract the code from the binary and then apply auditing techniques to find possible vulnerabilities.

Active Testing

The active testing procedure creates different tests in order to evaluate features compliance, performance and robustness of the application under test. Different techniques may be associated to this routine depending on the availability of the source code as for instance white box, grey box and black box testing which will be described in more detail in Chapter 4.

2.4 System Checking

From the network administrator security point of view it is important to assess the vulnerabilities of the managed system. Once the inventory of services, equipments and vulnerabilities are recognized, there are several actions which must follow. First, a check if the vulnerabilities are exploitable in the current deployment is required. Then, assuming that they are, a check if correction patches or mitigations already exist must be performed. Finally, if the services remain vulnerable, the administrator can take the necessary measures to make a trade off between the desired level of security and available flexibility to existing services.

2.4.1 Risk Measurement

Different approaches are found in the literature [16, 18] which focus on measuring the risk exposition of a network. Common metrics are computed using existing, historical and probabilistic vulnerability measurements. The exposure risks depend of the vulnerability successfulness, propagation and remediation level. Indeed, risk measurement has been represented using attacks trees, graphs and Petri nets [45, 47, 65, 72, 88, 101, 113, 122, 126, 138]. These formalizations supplement the risk exposure with information about the relationship between the vulnerabilities and the network architecture. Their main objective is to help assessment testers to visualize and analyze complex attacks.

Attack Trees

Attack trees [122] were designed to model security threats. Attacks trees are a special case of Fault Trees [140] where their goal is rather to formalize the tolerance of engeneering faults. They summarize possible actions or paths that must be taken to perform an assessment test.

This formalization represents each action of the attack as a node of a tree, where the root node represents the desired result actions. An action represented by an internal node is considered to be concluded when one or all of its children (depending if defined as a disjunction or conjunction respectively) are satisfied, i.e. a bottom-up flow. Thus, leaf nodes represent independent actions that should be performed. To differentiate a disjunction from a conjunction of actions in an internal node, the "and" string is attached to the bottom of the internal node if the action is a conjunction, and an empty string where there is a disjunction. Figure 2.2 illustrates a social threat to obtain access to the network.

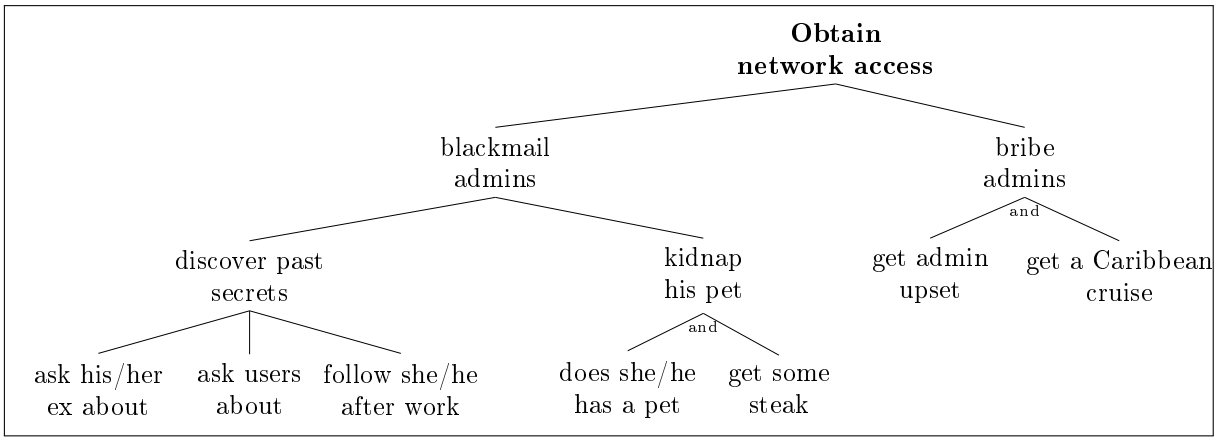


Figure 2.2: Attack tree example

Attack Graphs

Attack graphs [45, 46, 65, 72, 88, 113, 138] are based on a similar concepts, however this type of representation allows to model cycles in the attacks. Each node in the graph represents one goal or status in the attack while the edges represent condition in the transitions between the goals. In practice, generating an attack graph manually is a tedious and error prone task. Therefore, the authors of [126] proposed an automated generation of attack graphs able to help deciding which are the most cost/efficient attacks.

Figure 2.3 illustrates an equivalent social threat as depicted by Attack Trees but using Attack Graphs.

Attack Nets

M. Dacier et al. [46] introduce a mathematical model using Markov Chains to evaluate the mean effort need by an attacker to send the system into a security failure state. The approach uses Petri Nets [112] to build a model which will allows to characterize and evaluate the system security. Some approaches have extended the attack modeling to colored or stochastic Petri net attacks [47, 48, 101].

In the simplest attack nets approach, the goals are represented in a Petri net as places, while each transaction defines the action that the attacker has to perform in order to pass to the next goal. Contrarily to attack trees/graphs, attack nets allow to model concurrent behavior since the tokens represent the current state in the attack and each movement defines the steps that it has to follow to satisfy the goal. Figure 2.4 illustrates an equivalent threat as depicted by Attack Trees and Attack Graphs but using Attack Nets.

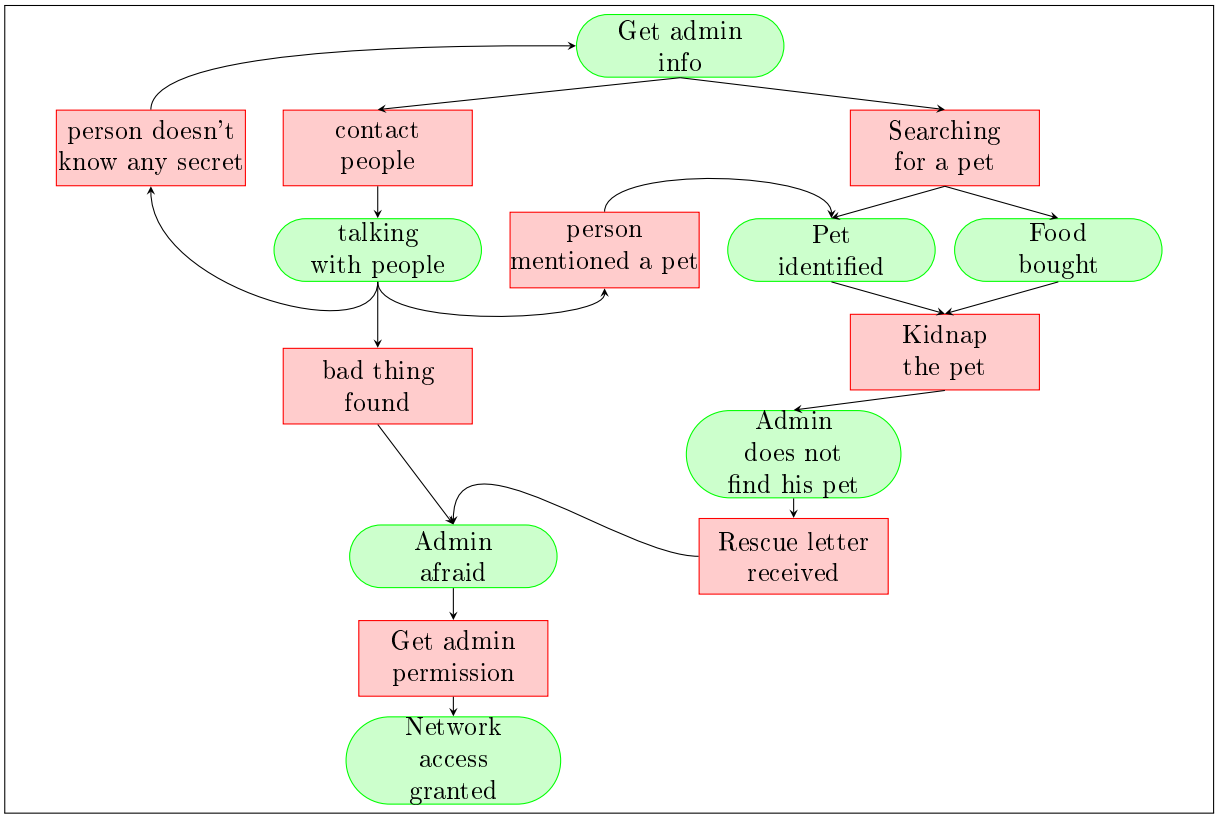


Figure 2.3: Attack graph example

2.4.2 Access Control Mechanisms & Traffic Inspection

Firewalls and Intrusion Detection Systems allow administrators to filter, analyze and log all traffic in the network. Filtering works on the basis of known patterns or learnt rules which may exploit specific vulnerabilities. However, for the same vulnerability, several exploits may exist, thus patterns for the exploits are not always identical thus bypassing filtered patterns. In contrast, the analysis of traffic can be used to trigger alarms when abnormal traffic is observed. Finally, logging may be useful to understand attacks that were not detected in the network.

Following the real-time intrusion detection model introduced by D. Denning in [52], the abnormal patterns searched on the network traffic are described below:

- Break-in attempts (for instance multiple telnet and user/passwords)
- Masquerading (login as someone else, but perform different operations)
- Privilege escalation by legitimate user (becoming root by a regular user)
- User leakage (legitimate user, connecting at non allowed times)
- Inference by user (get more data than needed)
- Malware (Trojan, Virus, Worms)
- Denial of service (resource monopolisation: bandwidth, CPU, etc)

Indeed, the model is composed of different components which will be used to identify and filter such patterns (assuming that security violations can be detected by monitoring the audit records):

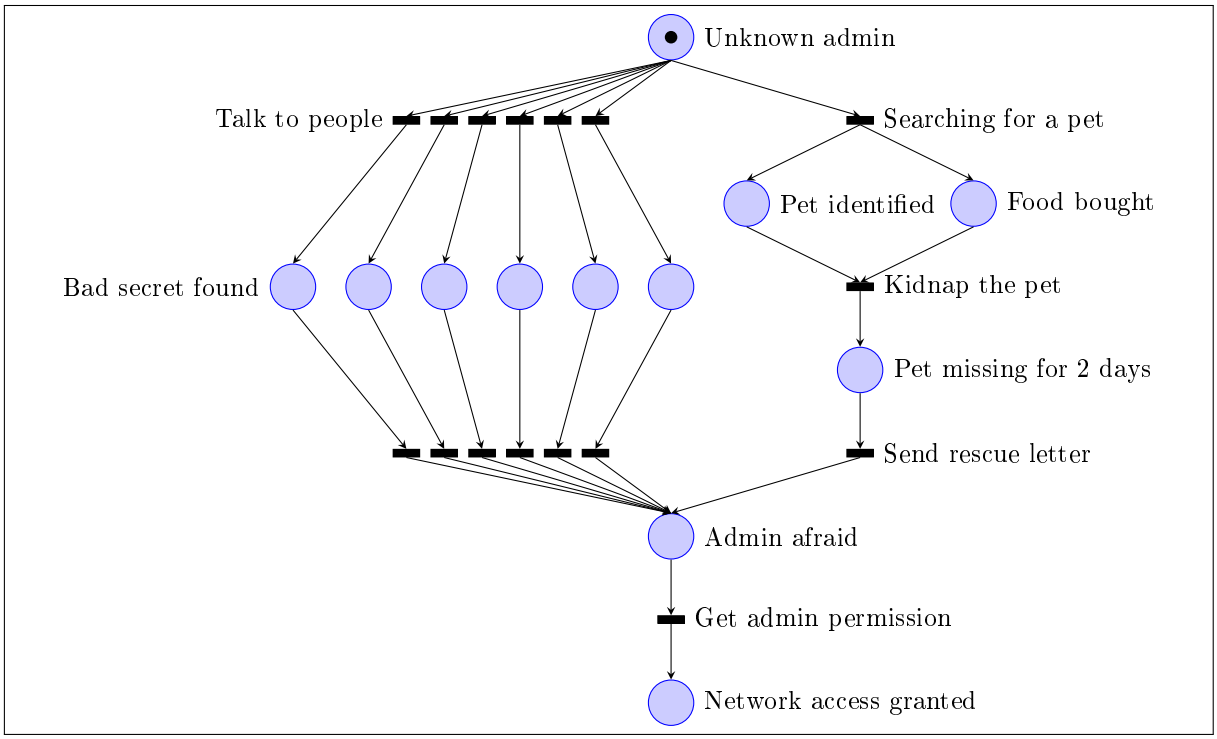


Figure 2.4: Attack net example

- Subjects (can be a user/program/system)
- Objects (resources: files, commands, devices or users)
- Audit records (system generated in response to actions)
- Profiles (statistical description of the subject-object interaction)
- Anomaly records (generated when anomalies are detected)
- Activity rules (actions to be taken when a condition is satisfied)

2.5 Summary

Assessment is the first step to take in order to protect a network. Since running services may not always be secured or probably just their interaction is not. A security breach can be opened in the network and thus jeopardize its whole behavior. It is expected that the security group acts like real attackers but avoiding exposing the integrity of the system to a vulnerable state. The first contribution of this thesis (chapter 5) proposes an environment where different scanners and fingerprinting tools can interact in the construction of a general information model for services that directly or indirectly interact with VoIP devices. Once that information model is constructed, the assessment group may use it to feed different attacking tools and techniques.

Chapter 3

Fingerprinting

The process of remote fingerprinting refers to the identification of a specific software or protocol based on the observed network traffic. Although, assuming that standardized behavior should be implemented in devices and applications, subtle differences coming from minor details not fully specified, different interpretations of the specification, different set of supported features or implementation bugs may allow to identify and to discriminate the origin of a given traffic.

Fingerprinting became an important methodology among the practices of security professionals. It started with the pioneering work of Comer and Lin [54]. Fingerprinting approaches are not 100% precise, however their accuracy increases with the amount of information obtained and analyzed from the sources. Although, it is important to note that they can be deliberately misled, information coming from a source can be intentionally scrubbed - maliciously or not [76, 99, 129, 139]. Therefore, the difficulty to conceive such attacks/defenses is a challenge based on the quantity of signatures identified by the fingerprinting system in its analysis of the hosts.

We understand by signature the characteristics which differentiate messages belonging to one device from the rest. Those are the characteristics found in the properties of the messages that are always the same or maintain a value related to a specific function. Related to this concept are the features of the system. Features are considered as the set of fields identified as signatures associated with the value for each of the known devices. For instance, if we define:

$System\_Features =< feature_0, \ldots, feature_n >$

where each feature is linked to a specific characteristic of the message and these are of the form

$feature_i = signature_0, \ldots, signature_{\#devices}$

then, each signature is the value for the i-device.

3.1 Fingerprinting Classification

In the last decade several approaches have presented different methods to identify the hosts or software existing in a network. This classification is often performed based on network messages generated by the hosts. These approaches, can be divided in three families:

1. those that are actively involved in the flow of messages, therefore they classify a host based on its replies to messages generated from the fingerprinting system,

2. those which passively observe the messages on the network and one by one observe their properties to build a classification and

3. those that once the fingerprinted host initiates a connection, then the fingerprinting system has the ability to interact with.

3.1.1 Active Fingerprinting

Active fingerprinting was first introduced in 1994 [54]. Through the injection of crafted/normal network packets, the active fingerprinting uses many elements to discover the operating system of a remote device under investigation: version, services, open ports, etc. The evaluation of these replies provides a rich set of information that can be useful for detecting the different protocol stack implementations. The system looks into several fields of the replies, for instance, reply type, reply errors, specific contents, retransmission delay, etc.

To be successful, the fingerprinting system has to find the set of queries that generates different responses in each different type of software and then creates a database of queries/signatures. Once the system proceeds to fingerprint a device, a subset of the recognized queries is sent to the target and, based on the replies from the device, the fingerprinting system should be able to guess the software running in the target.

Active fingerprinting techniques are quite efficient and can help to detect the topology and running services in which a small/medium network is composed. However, it can be disruptive since most of the sent packets violate the specifications of given network protocols.

3.1.2 Passive Fingerprinting

In 1997 passive fingerprinting was first introduced [110]. In contrast to active fingerprinting, passive fingerprinting is not meant to be invasive, it just monitors the traffic of the network and processes the observed messages to identify the source implementation and current state in the protocol.

Using passive discovery techniques has many advantages as described by O. Arkin in [19]. For instance, the monitoring of active items in the network, no impact on the performance of the network and detection of elements which are behind firewalls. However, it also presents weaknesses which are basically related to "what you see is what you get". Thus, the deployment location will restrict the flow visibility only to the partial set of services which are actively working in the network.

Not everything can be determined passively; as a consequence, services or host classification should be extended and compared with other types of analysis.

Passive fingerprinting, however, could be more challenging since the it does not have the possibility of intercept/modify in anyway in the flow of the network as an active approach does.

3.1.3 Semi-Passive Fingerprinting

Semi-passive fingerprinting was introduced in 2005 by T. Kohno et al. in [86]. The fundamentals behind this technique is that the fingerprinting system is not allowed to initiate a session, as it is the case for passive fingerprinting. However, once a session has been established by the host, the fingerprinting system can modify the generated replies in order to trigger events in the following of the session. This technique assumes that the fingerprinting activity takes place in between the session, either as man in the middle (e.g. an Internet Service Provider (ISP), Intrusion Detection System (IDS), attacker) or as one end point of the session (e.g. the same webserver to whom the request is generated). Thus, this family is more restricted than passive fingerprinting since it can fingerprint active hosts which only send traffic through its scope. However, it can trigger specific actions in the target host which will allow its recognition, like the procedures in the active fingerprinting family.

Indeed, it can fingerprinting hosts behind a NAT or firewalls since it only waits for the host to make the connection.

3.2 Packet Fingerprinting

Independently of the active, passive and semi-passive families, fingerprinting techniques can be also based on properties of the packet or in protocol level information (i.e. state machine of the implementations). These methodologies have their pros and cons, and they are explained in detail in the following subsections.

3.2.1 Packet Fingerprinting

Packet fingerprinting techniques consider the message as the whole source for analyses.The typical signatures search are based on the information obtained from the observed messages, for instance, banners, capabilities and flags, content order, etc.

For instance, figures 3.1 and 3.2 show four SIP messages (two messages each figure) generated from two different sources illustrating such subtle differences between implementations. First, it can be observed that in *Equipment A*, despite any possible configuration, all messages generated by such device share specific properties (highlighted in the corresponding figure):

- ordering of Message Headers (e.g. Via before Max-Forwards, Max-Forwards before To and so on),
- length of the first section of the Call-ID (note that after the "@" the device writes the local IP/URL),
- order in which the Allowed Method are specified,
- the banner written in the User-Agent Header, etc.

For the messages of figure 3.2, the fields call-ID, allowed methods and banner have the same values for all messages generated from *Equipment B*. However, they are different with respect to the *Equipment A*. One exception is the Message Header order, in which the headers Expires and Content-Length do not maintain a strict order (i.e. they had been switched).

SIP Equipment A

```
REGISTER sip:192.168.1.144 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:7060;rport;branch=z9hG4bKgydxyvae
Max-Forwards: 70
To: "humbol" <sip:5555@192.168.1.144>
From: "humbol" <sip:5555@192.168.1.144>;tag=jygph
Call-ID: ibfvgflwrrpzqbe@192.168.1.2
CSeq: 928 REGISTER
Allow: INVITE,ACK,BYE,CANCEL,OPTIONS,PRACK,REFER,NOTIFY,INFO
Contact: <sip:5555@192.168.1.2:7060>;expires=3600
User-Agent: Twinkle/1.0.1
Content-Length: 0
```

```
INVITE sip:79401@192.168.1.144 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.49;rport;branch=z9hG4bKomjgpxec
Max-Forwards: 70
To: <sip:79401@192.168.1.144>
From: "Bob" <sip:6666@192.168.1.144>;tag=nsxsr
Call-ID: tjqbyxvysbcramy@192.168.1.49
CSeq: 729 INVITE
Allow: INVITE,ACK,BYE,CANCEL,OPTIONS,PRACK,REFER,NOTIFY,INFO
Contact: <sip:6666@192.168.1.49>
Content-Type: application/sdp
Supported: replaces,norefersub,100rel
User-Agent: Twinkle/1.0.1
Content-Length: 304
```

Figure 3.1: Equipment A: message signatures

Based on those visually identifiable characteristics, one can make a classification of the source entity from the new messages. Table 3.1 depicts these signatures that were observed purely from a syntactic point of view.

SIP Equipment B

```
REGISTER sip:192.168.1.144 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.20:5060;branch=z9hG4bK4205b326
From: <sip:7940@192.168.1.144>;tag=000b46d9cb-1a84cfd8
To: <sip:7940@192.168.1.144>
Call-ID: 000b46d9-cb860003-66d2804f-527006cb@192.168.1.20
Max-Forwards: 70
CSeq: 102 REGISTER
User-Agent: Cisco-CP7940G/8.0
Contact: <sip:7940@192.168.1.20:5060;transport=udp>;
Content-Length: 0
Expires: 3600
```

```
INVITE sip:611@192.168.1.144 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.49:5060;branch=z9hG4bK50979e8b
From: "6666" <sip:6666@192.168.1.144>;tag=001ae2bc8b-4f6a3bc6
To: <sip:611@192.168.1.144>
Call-ID: 001ae2bc-8b7c001a-40b4297e-1611ee91@192.168.1.49
Max-Forwards: 70
CSeq: 102 INVITE
User-Agent: Cisco-CP7940G/8.0
Contact: <sip:6666@192.168.1.49:5060;transport=udp>
Expires: 180
Allow: ACK,BYE,CANCEL,INVITE,NOTIFY,OPTIONS,REGISTER,UPDATE
Supported: replaces,join,norefersub
Content-Length: 276
```

Figure 3.2: Equipment B: message signatures

	Equipment A	Equipment B
Call-ID Length	15	35
Allow Order	INVITE,ACK,BYE,CANCEL,OPTIONS, PRACK,REFER,NOTIFY,INFO	ACK,BYE,CANCEL,INVITE,NOTIFY, OPTIONS,REGISTER,UPDATE
User-Agent Banner	Twinkle/1.0.1	Cisco-CP7940G/8.0

Table 3.1: Localized signatures from equipment A and B

3.2.2 Protocol Level Fingerprinting

Protocol level fingerprinting focuses its analysis on the behavior observed from the target entity. The typical signatures search for this category are based on the type of responses to specific events, time delay within request/responses, underlying sequencing function, etc. G. Shu et al. introduced in [127] a formal approach to analyse protocol fingerprints. It is based in a Parametric Extended Finite State Machine (PEFSM), for which each transaction in the state machine represents a packet modeling the critical content of it (e.g. sequence number). Similar approaches Fingerprinting techniques bellowing to this category can be modeled and compared using this method. Indeed, the authors argued that longer fingerprinting sequences can be identified thanks to the automatic analysis nature of the approach.

Recently, the work by J. Caballero et al. [37] described a novel approach for the automation of Active Fingerprint generation which resulted in a vast set of possible signatures. It is one of the few automatic approach found in the literature and it is based on finding a set of queries (automatically generated) that generate different responses in the different implementations.

3.3 Fingerprinting Applications

Fingerprinting applicability ranges from topology discovery, hosts identification, attacks, malwares & spam detection, forensic activities up to copyright infringements. Such applications are illustrated in figure 3.3 and described below.

3.3.1 Assessment

Some of the most well known network fingerprinting tasks are done by Nmap, using a set of rules and active probing techniques. Passive techniques became known mostly with the P0F tool,

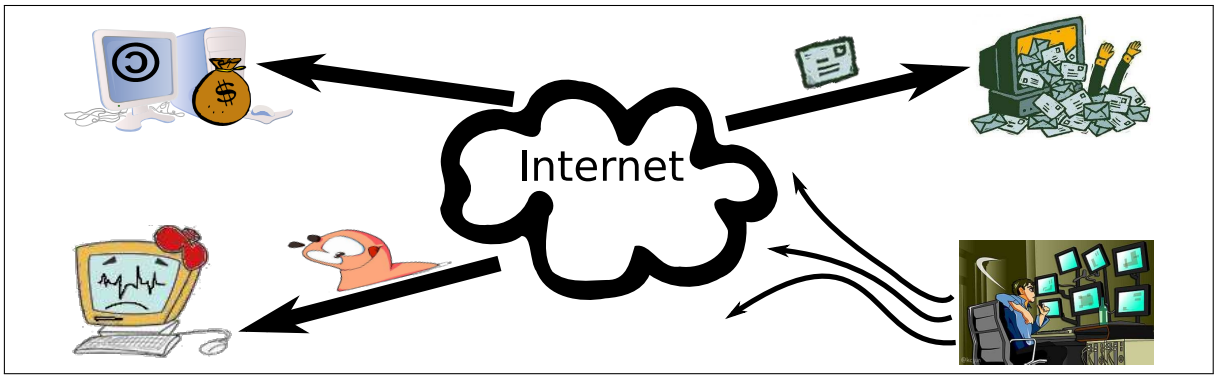


Figure 3.3: Fingerprinting Applications

which is capable to do OS fingerprinting without requiring active probes. Many other tools like (THC AMAP[18], XProbe[19], DISCO[20]) do implement similar schemes.

In security assessment tasks many operations rely on the precise identification of a remote device or a mere identification of part of it (e.g. network protocol stacks, services). Fingerprinting is essentially required for evaluating the security of a remote and unknown system. On the other hand, attackers can, as well, use this technique in order to identify hosts running vulnerable services.

Device Enumeration

In 2002, S. Bellovin proposed in [26] a technique to count the number of hosts that are hidden behind a NAT. This approach is based on the identification of the *id* field of an IP header. The id field, or mostly known as IPid field, is a string used to make packets unique. However, Bellovin found that most implementations implement such fields as a counter. Then, observing all the traffic forwarded by the NAT, one can make an estimation of how many active hosts are hidden behind the NAT.

T. Kohno et al. in [86] identified that hardware deviation of the system clock may be detectable from the timestamp of TCP packets, regardless of the remote location of the fingerprinting system. This technique not only allows to identify active hosts behind a NAT, but permits to distinguish virtual honeynets, virtual machines and real computers due to the difference of their clock oscillations.

OS Discovery

In 1994 Comer and Lin [54] first proposed an approach to probe differences between TCP implementations. Their technique consists in sending abnormal - invalid combination of flags - or malformed TCP messages to the host and observing the replies. Their original motivation was to find implementation flaws, to deduce characteristics of the design and to check protocol conformance in the implementations. Nmap developed by G. Lyon [97] was first released in 1997 and evolved to what became the preferred port scanner [96]. Thus, services which are running on each of the open ports can be actively fingerprinted by Nmap where for each of those service it may proceed to identify properties, running version, etc. Many other tools like (AMAP, XProbe) do implement similar schemes but they lack of precision and usability.

[18] `http://freeworld.thc.org/thc-amap/` last checked on December 2008
[19] `http://xprobe.sourceforge.net/` last checked on December 2008
[20] `http://www.altmode.com/disco/` last checked on December 2008

V. Paxson showed in 1997 in [110] the first passive approach able to distinguish different behavior from several TCP implementations. He based most of his observation on how hosts under congestion deal with updating window values, retransmission timeout range, behavior under out of sequence data, response delays, etc.

Service and Protocol Identification

Approaches like [28, 29, 42, 84] introduced classification methods based on headers only. The features analysed were related to host behavior, direction of flow, packet size/order/arrival, etc. Thus, these approaches need a complete (sometimes a partial prefix suffices) sequence of messages in the flow to be able to classify the running service. More important, all these methods rely on an automated mechanism to train their systems (e.g. Support vector machine (SVM), Gaussian-filtering, supervised machine learning).

Recently, work in the area of identifying properties of encrypted traffic has been reported in [30,144]. These two approaches used probabilistic techniques based on packet arrival interval, packet length and randomness in the encrypted bits to identify Skype[21] traffic or the language of conversation. While these works addressed the identification of the protocol building blocks or properties in their packets, they do not assume a known protocol and they did not propose an approach for identifying specific implementation stacks of the same protocol.

There have been recently similar efforts done in the research community with practical reverse engineering of proprietary protocols [66,67] and a simple application of bio-informatics inspired techniques to protocol analysis [24]. These initial ideas matured and several authors reported good results of sequence alignment techniques [43, 44, 70, 91, 108, 143]. Another major approach for the identification of the structure in protocol messages is to monitor the execution of an endpoint and identify the relevant fields using some tainted data [35,94].

Finally, other solutions have been proposed in the literature in this research landscape. Flow based identification has been reported in [74], while a grammar/probabilistic based approach is proposed in [31] and respectively in [98].

Application Identification

Application layer fingerprinting techniques, specifically for SIP, were first described in [73, 145]. These approaches use both active and passive fingerprinting techniques. Their common baseline is the lack of an automated approach for building the fingerprints and constructing the classification process. Furthermore, the number of signatures described are a minimal set of observed responses to unusual messages, banners and content presented in the message which let the systems easily exposed to approaches as the one described by D. Watson et al. [139], which can fool them by obfuscation of such "observable" signatures.

3.3.2 Analysis & Prevention

Intrusion Detection System may use fingerprinting approaches to detect rogue systems, stealth intruders and the behavior of worms which spread over the network. Filtering techniques, such as for spammer detection may use fingerprints to identify spam messages. For instance, syntax message fingerprinting can be used to filter abnormal traffic the classification is done per message. Indeed, the SIP [120] specification says:

> *"servers MUST NOT attempt to challenge CANCEL requests since these requests cannot be resubmitted. Generally, a CANCEL request SHOULD be accepted by a server if it comes from the same hop that sent the request being canceled"*

[21] `http://www.skype.com`

then an attacker can spoof a CANCEL message in the network and therefore teardown the session as illustrated by figure 3.4.

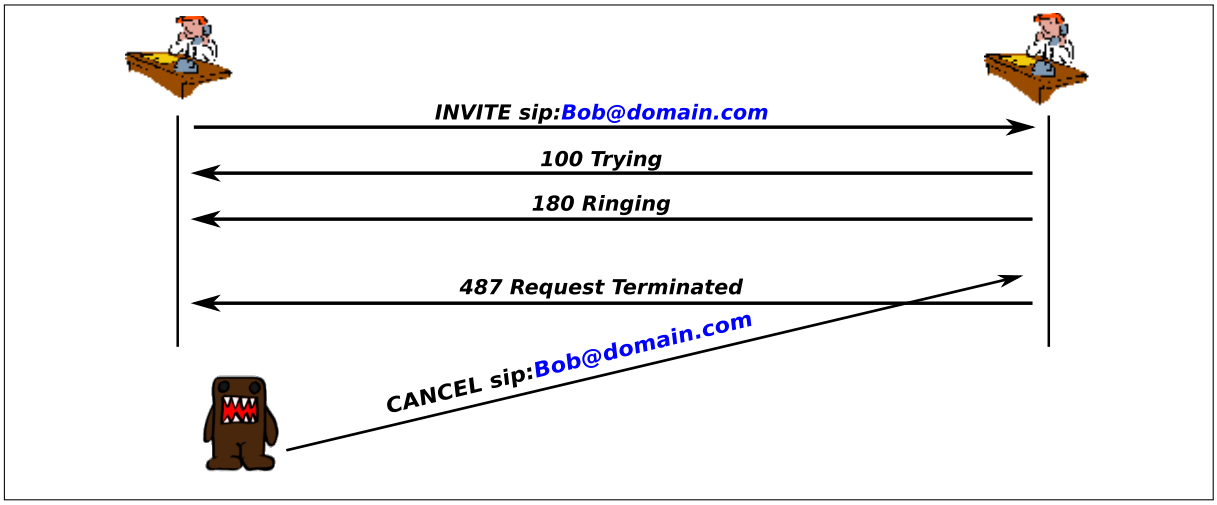


Figure 3.4: CANCEL tear down session

However, if an IDS detects that the CANCEL message do not match the signatures from such device, it can assume that the host has been impersonated and it could the message dropp to avoid the DoS attack.

Spam Detection

M. Chang and C. K. Poon [39] address an email SPAM detector trained from a collection of traces. This approach focuses on identifying human written sentences from spam generators. It applied techniques to analyze the lexicon of the messages and do not use information obtained from the protocol.

Attack Analysis

C. Leita et al. introduced in [91] a technique capable of fingerprinting the expected behavior of services targeted by malwares. Almost simultaneously W. Cui et al. [44] proposed a similar technique. In both approaches the services behavior is analysed from stored traces having no knowledge of the underlying protocol. The system is then capable of simulating the targeted services as a typical honeypot. Thus the main objective is to fool malwares and during their interaction make a deeper analysis of the techniques employed by the malware.

Forrest et al. in [64] proposed an anomaly detection technique in which abnomal behavior is identified based on the system calls that it generates, since reverse engineering in the binaries is not always feasible.

3.3.3 Copyright Infringements

Another important applicability of fingerprinting system resides in blackbox devices/application testing for potential copyright infringements. In the latter case, when no access to source code is provided, the only hints that might detect a copyright infringement can be obtained by observing the network level traces and determine if a given copyright protected software/source code is used unlawfully.

3.4 Summary

Network fingerprinting is the main topic of many emerging research fields. Since several new malwares/attacks are developed or new protocol are being designed - either as standards or commercially - the need to automate the fingerprinting process is the target of several research teams. In this chapter, we discussed different techniques that have been used and how they evolved. However, we found that automatic techniques for fingerprinting the application level of the messages are missing . Chapter 6 describes a novel contribution where a passive application level fingerprinting is described. This approach tries to fill the gap of fingerprinting techniques which identify the application running based only in the messages generated. It is important to note that the classification model automatically identify signatures per message.

Chapter 4

Vulnerability Testing

Vulnerabilities represent a major issue in any networked application, especially in an emerging technology landscape like VoIP. The number of implementations is increasing with both the number of devices connected to the Internet and the number of software components being used to deliver services on the Internet.

For instance, rapid deployment of SIP enabled boxes opens new revenues for attackers: they can now use the Internet to carry out new fraud schemes against telecommunication providers. Exploiting these new fraudulent schemes can not only lead to substantial benefits for the malicious users but also, in some cases, to more severe consequences in both enterprise and private networks (e.g. deactivation of emergency call capacity on a Voice over IP Phone).

Fault injection has been a major topic of research over the last decades [137], but very recently the term "fuzzing" cornered the art of injecting user supplied data to applications in order to detect potential vulnerabilities in software implementations. The most notorious exploitation are due to generic buffer overflows and format string exploitation, but higher level application and target specific vulnerabilities can also be discovered by fuzzing.

The conceptual idea behind fuzzing is very simple: generate random/malicious input data and inject it in an application. This approach is different from the well-established discipline of software testing where functional verification is checked. In fuzzing, this functional testing is marginal; much more relevant is the goal to rapidly find potential vulnerabilities. Protocol fuzzing is important for two main reasons. Firstly, having an automated approach eases the overall analysis process. Such a process is usually tedious and time consuming, requiring advanced knowledge in software debugging and reverse engineering. Second, there are many cases where no access to the source code/binaries is possible, and where a "black box" type of testing is the only viable solution. Therefore, major software vendors, ranging from Microsoft and Cisco, to Mozilla and Redhat have introduced fuzzing as a key phase in their software development life cycle [117, 132].

4.1 Origins of Vulnerabilities

Vulnerable software is usually coming from bad coding & design techniques or even from common errors during the implementation phase among other things. Typical exploitable errors come from memory access violation, improper input data validation or wrongly computed logic assertions.

Memory Overflows

Programming languages like C allow the user to have full control over the allocation of memory. If the allocation control fails due to improper coding, normal execution can be violated and then

the system may be exposed to exploit attacks [20].

Figure 4.1 illustrates a snipped of C code where data is copied (in line 6) to a buffer of smaller size without boundary checks. Thus, adjacent data in the buffer is modified allowing to change the normal flow of the program.

```
1  #include <string.h>
2
3  void bof_function(char *str)
4  {
5          char buffer[16];
6          strcpy(buffer, str);
7  }
8
9  void main()
10 {
11         char large_buffer[256];
12
13         memset(large_buffer, 'A', 255);
14         bof_function(large_buffer);
15 }
```

Figure 4.1: C code overflow

Format Strings

String format functions in the C language take a string format followed by the arguments. The string format can contain either printable characters or conversion specifications. The conversion specifications fetch zero or more arguments, for which its value will be replaced. If the arguments given to the string format function are different with respect to the quantity of conversion specifications, then major exploitation opportunities exist [92]. In the case of an overflow, the string format function will grab the next value in the memory. If the *%n* conversion is used, information can be printed in the memory to deviate the normal flow of the application. Figure 4.2 shows a vulnerable C code where if the user calls it with the *%x* argument, memory information extracted from the stack will be printed.

```
1  void main(int argc, char * argv)
2  {
3          printf(argv[0]);
4  }
```

Figure 4.2: C format string overflow

SQL Injection

SQL injections occur when applications do not correctly escape input values and then these values are been used for interacting with relational databases [95]. The following statement illustrates a typical SQL query:

```
query = "SELECT * from User_Table where username = '" + input_username + "';"
```

where if the input_username is *Alice*, for instance, it will search for all the registers in the *User_ Table* named *Alice*. However, if the *input_ username* is:

```
Alice'; DROP TABLE User_Table;#
```

it will make the same query, but just afterwards it will execute a new statement (DROP) and remove the whole list of users.

Cross-Site Scripting (XSS)

Cross-Site Scripting is another type of code injection which allows an attacker to execute malicious code in the client-side application across a non malicious third party [125]. This type of attack is found in web applications which offer the user inputs to other visitors without filtering. Assuming a webpage that displays the user input without filtering and that the input is in fact a script code like:

```
<script> alert('XSS attack!!'); </script>
```

thus, the displayed information will be emptied but JavaScript code will be executed. Executing malicious JavaScript code can lead to a full compromise of the host computer. For instance, MPack is a publicly known malicious tool sold in the black market which uses this type of vulnerability. If a host visits a compromised webpage, MPack sends a script code able to identify the host and therefore determine if any vulnerability exists on it [89]. If the host contains a vulnerability known by MPack then the corresponding exploit is used to compromised the system.

XSS can be classified into two types [125]: reflected and stored. In the first one, the executed script is included in the request itself, therefore the attacker has to send the request to the victim; for instance by email. The stored XSS vulnerability is more serious since the script is stored in a permanent repository of the server and any user visiting the webpage will execute this malicious script.

Race Conditions

Race conditions usually appears in muti-thread applications in which the timing of operations execution is highly critical. A typical scenario is the case where an application check the rights to do certain procedures and in the meantime (caused by external factors) the predicates is no valuable anymore. Causing the application to perform an action for which it did not have the rights anymore. Thus, the problems can be related to deadlock, livelocks, locking failures or interference caused by untrusted processes [141].

4.2 Whitebox, Greybox and Blackbox Testing

As fuzzing is one more discipline of software testing, the high level definition of the approaches used for testing is maintained [25]: Whitebox, Greybox and Blackbox testing.

In white-box testing, the source code of the application is analyzed in an attempt to track down defective or vulnerable lines of code. This includes manual inspection of source code as well as automated source code analysis using vulnerability scanners.

In black-box testing, the internal code of the application is not considered. Instead, the test is driven according to the requirements of the application; special input test cases are generated

and sent to the application. Then, the results returned by the application are analyzed for unexpected behavior that indicates errors or vulnerabilities.

Greybox testing has many of the advantages of the two previous techniques; it can generate input data based on observed analysis of the runtime code execution. Thus, the statistics used can help to improve the future inputs. This technique relies on the availability of the binary code, at least, and can be complemented with the protocol information of the application.

In practice, manual source code analysis is the prevalent approach for auditing and checking applications for the presence of security vulnerabilities. Manual analysis has the advantage that human analysts can understand the code and detect security vulnerabilities of any type. Unfortunately, there are also a number of significant drawbacks to this approach. One is that code auditing is a very time - and labor - intensive task. Since it requires well-trained experts to find vulnerabilities, this process is very expensive. Finally, while humans are very good at understanding programs, they can be tired, distracted, or losing focus. As a result, they might miss bugs, especially when these bugs are buried under thousands of lines of source code. Because of the difficulties and costs of manual code analysis, automated tools for finding vulnerabilities are desirable.

4.3 Fuzzing

The first known fuzzing case is documented in the early 90's, when B. Miller et al. [104] observed that random input did crash most Unix applications. In that particular case, these crashes were due to electrostatic fluctuations in a remote dial-up connection induced over the physical wire. This observation brought this specific type of fault injection technique to the attention of security researchers. For over two decades, some significant research activities were performed in the testing community in order to automate the process of functionally testing the correctness of software with respect to some well defined properties [17, 50, 79, 83, 100]. The key difference with traditional testing is that fuzzing attempts to generate inputs and test cases leading to the discovery of vulnerabilities, but many terms like blackbox fuzzing, whitebox fuzzing maintain their semantics.

In the beginning, fuzzers were simple random data generation frameworks used in a manual way by security researchers. These frameworks were suited to generate application specific input data, ranging over multiple input formats: image files, command line arguments, multimedia files, and network data. This changed over time [50], where both increased activities of leading software companies like Microsoft [79] and recent research papers [51,55,69] and books [117,131, 132] significantly raised the interest on the potential of fuzzing for detecting vulnerabilities in blackbox, whitebox and graybox testing types.

In the context of SIP software, most SIP stack developers use fuzzing techniques to check their code, but most of the used approaches seem to be highly manual. The well known team at Columbia University, lead by Prof. Henning Schulzrinne [142], has been performing research on VoIP vulnerabilities, but in their work they defined some static test cases that were used with an open source fuzzer tool.

4.4 Fuzzer Environment Classification

Different categories of fuzzers have already been discussed, however it depends in the chosen context to know which is the most effective approach for a specific task (e.g. what is needed to be fuzzed, what is known from the protocol, available budget). Indeed, fuzzing just generates "malicious" traffic but then an environment in which it is possible to observe the effectiveness of the traffic is needed.

4.4.1 Fuzzing Frameworks

Fuzzing frameworks are complex projects. Their objective is to provide a reusable and generic environment where new fuzzers techniques can be easily deployed. Current environment provides directives to interact with the target (e.g. sending/receiving traffic by the network, load files in the applications, interact with the application memory), to monitor the state and to feedback information from the tests. Among the most popular frameworks we find Sulley[22], Peach[23], Autodafe[24], BeSTORM[25].

4.4.2 Special-Purpose Fuzzers

Special purpose fuzzers focus their test on a specific protocol (proprietary or not). They are generally quite effective and they do not require much interaction with the user to launch a new test. However, they are difficult to extend current test cases as well to any other new protocol that they were not intended for from the beginning. The most popular special-purpose fuzzers are Codenomicon[26], Mu-4000[27] which are capable to fuzz over 130 and 50 protocols respectively.

4.4.3 General-Purpose Fuzzers

General purpose fuzzers are designed to adapt the testing to the protocol being fuzzed. In many cases some information has to be provided to guide the generation of messages. While in other approaches, they could observe the traffic and determine, by different metrics, which fields they can modify in order to send the new crafted message to the target. Among the general-purpose fuzzer we can find ProxyFuzz[28], SPIKE[29].

4.5 Input Generation

> *"The input should in most cases be good enough so applications will assume it's valid input, but at the same time be broken enough so that parsing done on this input will fail."*
>
> Ilja van Sprundel [136]

The main challenge for syntax fuzzers is to generate the crafted message provided as input to the fuzzed system. Much more effort has been put in the quest for smart approaches for such generation. Basically, they can generate messages either from scratch or by mutating parts of an existing one. Mutation is often simpler since there may not be actual knowledge of the current protocol to flip a few bits. However, random fuzzers which do not use any knowledge of the underlying protocol are often less effective.

In contrast, generation from scratch must have certain clues about how it must proceed to create a single message, otherwise there is an extremely low chance to generate messages that pass the lexer (e.g. been discarded because some mandatory fields are missing). Usually, specifications taken from the protocol are used to manually construct such guidelines. C. Miller [105] made a

[22] `http://www.fuzzing.org/` last checked on December 2008
[23] `http://peachfuzzer.com/` last checked on December 2008
[24] `http://autodafe.sourceforge.net/` last checked on December 2008
[25] `http://www.beyondsecurity.com/` last checked on December 2008
[26] `http://www.codenomicon.com/` last checked on December 2008
[27] `http://www.mudynamics.com/products/mu-4000.html` last checked on December 2008
[28] `http://theartoffuzzing.com/` last checked on December 2008
[29] `http://www.immunitysec.com/resources-freesoftware.shtml` last checked on December 2008

survey of Mutation vs. Generation-Based fuzzing where he states that a generated-based fuzzer can execute 76% more code than with mutation-based fuzzer. A generation-based approach must create the message from scratch, thus messages can miss or have redundancy of some arguments. Also, it should be considered in terms of message diversity, that selection proportionally depends in the specifications given in the guidelines. Indeed, mutation techniques will consequently modify minor sections of the message maintaining a structure similar to the original message.

Syntax fuzzers can have an engine independent of the environment in which they are running, then the message creation can be sub-classified in Random, Block-based, Dynamic Generation and Grammar-based fuzzers [132] depending on the protocol knowledge and techniques used by the fuzzing guidelines.

Random Fuzzers

Random generators are the most basic fuzzers. They represent the very beginning of fuzzing and their techniques were able to discover numerous vulnerabilities, albeit their approach was naive. Basically, this technique treats a message as a random sequence of bits or bytes - as seen in figure 4.3 - which will be delivered directly to the testing unit.

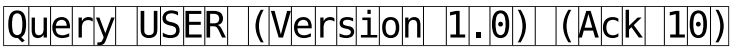


Figure 4.3: Random fuzzer message interpretation

Fuzzers of this type do not have any knowledge of the protocol and are, at most, suitable only for request/response protocols or file fuzzing.

Block-Based Fuzzers

Block-based fuzzers were first introduced by D. Aitel [17]. He pointed out that there are known factors in a network protocol and therefore, one can delimit the effect of unknown factors. In this way, the input space can be drastically reduced to only a subset of chosen parameters. Basically, this technique fragments the message in blocks; each block may represent fixed strings (these portions of the protocol that are static), random inputs (e.g. string, numbers, binaries) or a programmatic function (i.e. functions able to calculate lengths, checksum, etc). Thus, if the length of the inputs values from the generated message has been modified, the content length can be re-computed to remain correct.

Figure 4.4 illustrates how the same message than the one from figure 4.3 can be viewed from a block-based approach.

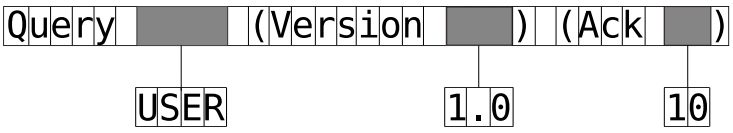


Figure 4.4: Block-based fuzzer message interpretation

Most current fuzzing approaches are based on this methodology[30]. Among them we find

[30] http://www.fuzzing.org/fuzzing-software and
http://www.fuzz-test.com/tools/ last checked on December 2008

SPIKE[31], Sulley[32], Peach[33], Autodafe[34] and many others.

Dynamic Generation

Dynamic fuzzers are not aware of the protocol being fuzzed but, however, they use heuristics, machine learning or evolutionary algorithms to induce the structure of the message. Thus, these fuzzers have a learning curve, in which if their attributes are well specified, the effectiveness of the fuzzing can increase based on feedback inputs or behavior analysis. Among the fuzzers of this type we find GPF[35], Sidewinder [57].

Grammar Based Generation

Input data validation ideas based on grammar formats dates back to the early 80s [25]. In this methodology B. Beizer proposes that every input must conform to a format, therefore he suggests the Backus-Naur Form (BNF) as a context-free grammar. Then, certain guidelines can be followed to use the grammar and to generate the messages, inserting subtle modifications which will generate malformation into the message. Certainly, grammar based methods were considered as fault injection since they generate the messages based on a valid specification. However, the work done by R. Kaksonen [83] was the first to take a grammar based message generation in the fuzzing context. This work represented the basis of the well known Mini-Simulation Toolkit from PROTOS.

Grammar based fuzzers are defined using context-free grammar. Their messages are created by the reduction of rules defined in the grammar. Eventually, a rule may be reduced to a function that may compute its value from a segment of the message (e.g. checksums, content lengths). Figure 4.5 illustrates the granularity in which a grammar based approach can represent a message being fuzzed. It can be interpreted as a tree representation induced from message based on a free context grammar, where the representation is a top-down left-right reduction of the items in the grammar and the leafs represent the content of the message.

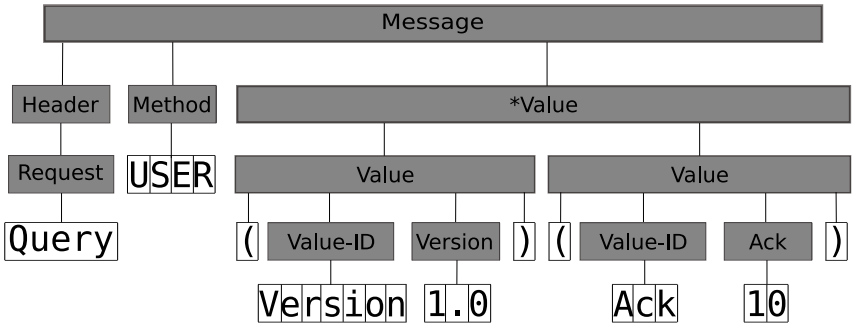


Figure 4.5: Grammar based fuzzer

[31] http://www.immunitysec.com/resources-freesoftware.shtml last checked on December 2008
[32] http://www.fuzzing.org/ last checked on December 2008
[33] http://peachfuzzer.com/ last checked on December 2008
[34] http://autodafe.sourceforge.net/ last checked on December 2008
[35] http://www.vdalabs.com/tools/efs_gpf.html last checked on December 2008

4.6 Advanced Techniques for Fuzzing

> *"Program testing can be used to show the presence of bugs, but never to show their absence!"*
>
> Edsger W. Dijkstra[36]

At the beginning, fuzzing approaches did look more like a random walk in a search space [104]. The search space is the sequence on the input data and the objective of its search is to find the inputs that lead to the discovery of one or several vulnerabilities. There are no standard metrics to evaluate the effectiveness of a fuzzer, however the basic ones relate to the speed to find unknown or known vulnerabilities or to the number of substitution of strings based on the values contained in a library for each of the fuzzed variables. Obviously, these types of metrics are not valuable in term of effectiveness. For instance, which fuzzer will be better? one that finds only one type of vulnerability as soon as it is launched or one that takes significantly longer but is able to discover a wider range of vulnerabilities types.

In the following sections, further research in the topic will be briefly described. These approaches have proposed search space explorations techniques such as code coverage tools, feedback, statefulness among others to improve their efficiency. Efficiency is associated with rapidly detecting highly relevant zones in the attack surface space and knowing when to stop a fuzzing process. In fact, the attack surface is that portion of the code that is reachable by the the external data. These advanced methods seek to test as much code as possible, therefore trying to modify entries to execute sections of the software which are less frequently used (e.g. deprecated libraries, proprietary features). They assume that these libraries were subject to less intensive testing and then possible some flaws may arise. This heuristic may take into account the impact that a given input (generated by the fuzzer) has on the target.

4.6.1 Stateful Fuzzing

Several applications rely on protocols which are session based, thus each of their current action will be reflected in the future of the session (i.e. they are stateful). Therefore, some of the information currently sent to it may not be immediately used but needed for generating a future request. Assuming that some specific information can trigger certain flaws, but only if used, we give a concrete example.

For instance, the Contact header of a SIP message is a mandatory header. It is used to know the location from where the remote entity can be contacted. During a SIP transaction, implementations just reply to the message in the currently open channel (i.e. the same port and address from where the message was received). However, there are cases where one of the endpoints needs to initiate a new request - or transaction - then there is a need to create a new channel. Thus, the new channel will be created using the information obtained from the Contact header. So, if the Contact header was initially crafted and improperly filtered by the application, the flow won't be triggered until the implementation needs to generate the new request.

Usually most fuzzers don't consider such statement and they send one message after the other in order to observe how the target unit responds to them individually. Indeed, there are no links between the sequence of sent messages (i.e. they are stateless). Few stateful approaches can be found in the literature, among them approaches like Mini-Simulation Toolkit [83] and Bug Catcher [61]. The later describes a simplistic method where message transitions are described using a context-free grammar. This approach is more suitable for reaching a context in the protocol rather than fuzzing (e.g. authenticating) and afterwards applying the fuzzing techniques.

[36] Source: Notes On Structured Programming, 1970, section 3, On The Reliability of Mechanisms.

Indeed, they won't be able to keep track of messages that are out of the session order, which in fact could as well raise possible vulnerabilities.

Interstate [77] and Snooze [22] propose a follow up list of messages described in a meta-language which defines the sequence of messages which has to be sent during the test. However, this list of messages is rather naive giving a weak sense of statefulness.

In contrast, some interesting work has been developed by D. Lee et al. [90], where they construct an automaton to induce from the ongoing messages in the network, the current state of the devices. A different approach has been taken by E. Bayse et al. [23] where the construction of the state machine is based on the protocol specification. The novelty is based in extracting a set of properties from the state machine specification, called invariants. Thus, these invariants describe the most important properties implementation behavior. However, the approaches [23, 90] analyses the state machine of the application focusing only for passive testing. Y. Hsu et al. [80] extended this approach to induce the state machine of the protocol - based on previous traces - in order to simulate the behavior of the protocol to be fuzzed, however this approach does not have any knowledge in the protocol and many subtle details can not easily be discovered. Thus, making it not really suitable for complex protocols (like SIP).

4.6.2 Unknown Protocol Fuzzing

Sometimes, the specifications of a protocol are not available, or too complex in order to take them into account when designing a specific fuzzer. In many cases, obtaining these specifications is an impossible process, as it was the case of the SAMBA project which took 12 years to fully reverse engineer the Microsoft SMB protocol [134].

Automatic protocol reverse engineering is a complex and difficult problem. It can target different levels. In the simplest case, the analysis only examines a single message. Such approaches are typically deployed as proxies which are kept in middle of the traffic of two normal endpoints and mutate random bits in the network flow[37]. A more general approach considers a set of messages of a particular type. An analysis process at this level would produce a message format specification that can include optional fields or alternative structures for parts of the message. Typically, the message semantics inference is addressed by bio-informatics algorithms (e.g. alignment) by approaches like the Protocol Informatics Project (PI) [24]. Finally, in the most general case, the analysis process operates on complete application sessions. In this case, it is not sufficient to only extract message format specifications, but also to identify the protocol state machine. Moreover, before individual message formats can be extracted, it is necessary to distinguish between messages of different types. Different state machine inference algorithms have been proposed [44, 80, 91, 108], all of them based on observed traces. However only [80] used the inferred state machine for doing fuzzing while the other two used it for simulating the behavior of the observed protocol.

4.6.3 Feedback Fuzzing

Feedback gathered from the assessed platform can be used to evaluate and drive the fuzzing process. It is important during a fuzzing process to cover as much as possible executed code in a target application and to be able to fine-tune and learn when and how to shift the fuzzing process. It is also essential to know, in a fuzzing scenario, when to stop because no other step can provide any valuable advance. A new research line focusing on the generation of smart fuzzing approaches, capable of assessing their impact in terms of code coverage, memory impact and capable of evolving during a life-cycle is described below.

[37] `http://www.theartoffuzzing.com` last checked on December 2008

The search space for fuzzing is quite big depending on the type of protocol being tested. Assuming a simple snippet of a C program as shown in Figure 4.6, the integer entry *value* may have 2^{32} possible values for which only one (value "3") will execute a new section of the code. In such terms, it is highly improbable for a fuzzer to pick the "magic" value.

```
1 def check( value ):
2     if value == 3:
3         readData()
4     else:
5         continue
```

Figure 4.6: C code snipped

As it is not always possible to get the source code of the application, the methodologies **Whitebox** and **Graybox** fuzzing are named depending in if the source code or binary is available, respectively. In current blackbox fuzzing, few feedback can be used to drive the fuzzing process rather than the replies or behavior from the application. Whitebox fuzzing can leverage code path related information to generate feedback for a fuzzer and to shift its fuzzing engine [38,51,55,68,69]. Feedback should observe deviations in the target with respect to estimated behavior as a good indicator of the potential of the current input sequence. Also analysis of the code execution, memory operations and tainted data propagation monitored in a virtualized environment could help to improve the techniques. Whitebox and graybox activities can use this in order to make smart and intelligent fuzzers, capable to learn from their experience and perform efficient fuzzing. Then, the fuzzing process is in essence a two players repeated game, where the fuzzer is in fact playing a strategy with the target equipment.

Recent work [51] combined techniques on identifying segment blocks into the binaries reached by inputs that can be used together with evolutionary algorithms to improve the code coverage. Further on, [38] extended some ideas coming from symbolic computations in the automatic instrumentation of executable. Symbolic input is executed and traced by the proposed framework. This approach makes it possible to run a single input and explore multiple execution paths. It suffers from the drawback that access to the source code of the tested application is required. From this point of view, this approach is closer to static analysis code approach than to fuzzing. The level of instrumenting the target application might not be so strict and requires access to the original source code. Similarly, C. Pacheco et al. [109] test JDK libraries where they create a set of randomly generated sequences. Each sequence is linked to an existing method, and therefore in each new generation the method's arguments are filled with other sequences found in the set that matches their type. Metaphorically speaking, it creates puzzles pieces which may result in a program. However, sequences may be filtered based on their output execution and avoid duplicated structures. A more generic approach is described in [69], where constraint solving is the underlying model to generate input data. New input data is generated by running some initial seed data, analyzing the constraints encountered in the control flow of the program and inducing how the input data will satisfy the constraints; then the problem is reduced to a constraint optimization problem. One promising introduced idea there is that heuristics and more specifically generational search might be useful to explore the search space. Similarly to the previous approach, the model is whitebox oriented, but only access to the executed machine code is required. In [68] the authors extended their approach to also use grammar specifications to improve the seed for the whitebox approach. Thus, scoring the grammar token, the generated messages can evolve to cover a wider area of the source application. The authors of [55] couple a genetic algorithm inspired search model with a memory instrumentation technique in order to drive a fuzzing process. Indeed, they identify the flow of the program that is guided by tainted

data (i.e. variables that they had filled directly or indirectly by program inputs). Once identified the coverage of the tainted data, the flow of the application can be modified by changing the branch values in the machine code. This approach allows the system to check a wider area of code but it has many limitations, as for instance, the executed code may never be reached (e.g. if the machine code was modified just before the input data was filtered) and also lacks code tracking support in the target application.

4.7 Summary

Software flaws are more exposed than before since networks (e.g. Internet) are making those implementations reachable to almost everyone. Therefore, testing tools are highly used in the development process as well as afterwards by researchers and attackers. One discipline that lately got a lot of attention is fuzzing. From its naive beginning, it was able to find huge amounts of vulnerabilities, flaws and errors. As millions have been spent to repair damages from software attacks, security becomes a must rather to be considered as a feature. A lot of investment and research is on the way and new techniques are constantly been proposed. In the thesis, two novel contributions in the fuzzer context has been proposed, described in chapter 8. First, no current technique describes a full stateful testing. The few existing approaches address a vague definition of states, where everything has to be manually coded and there is no underlying checker to observe what could be abnormal traffic or not (related to the software state). Therefore, we address a stateful approach to test SIP implementations which merges active and passive testing in order to guide the test and evaluate it at the same time.

The second contribution improves the message syntax generation model. Advanced existing approaches modify syntax inputs by feedback from the source/machine code or through the use of a BNF to guide the construction. Either they construct a message from scratch (those who used a BNF) or mutate an existing seed to obtain derivations from the message (feedback oriented). However, no-one considered an approach where mutations can be based on BNF grammars using statistic measured from existing traces. Indeed, those statistics may allow to scale to more complex BNF grammars and to provide a wider diversity of messages (still close to what could be acceptable parameters of a message). Thus, we address this issue where the main key to improve the message generation based on statistics is to has a syntax parser as well as a syntax fuzzer generator.

Part II

Assessment Environment for VoIP Services

Chapter 5

Assessment

The best approach to avoid intruders to get into the network is to have the latest software updates, restrict already known vulnerable services and the most important of all, periodically perform comprehensive assessment tests. For these tests, network devices and hosted services must be automatically discovered by the assessment team and once discovered the devices must be tested for potential vulnerabilities. This topic is the main theme of this chapter and derived in the following publication [9]. Our work was challenged by providing an automated approach capable to be accurate in the discovery of the VoIP infrastructure, perform user driven attacks and allow dynamic extension with security plugins. The major challenges were posed by the numerous types of services that must be checked, the security constraints that exist, and additional firewalls and NAT devices that do increase the difficulty of an accurate fingerprinting and discovery phase.

The three main challenges that an automated assessment must meet are:

1. Discover as much as possible the topology, the services and the configuration running on the devices in the network,
2. Store and provide all the information gathered in an easily usable manner,
3. Launch different assessment tests, discovery and/or attack actions using the information acquired.

It is worth noting that this approach is not meant to be worked out at once, but rather progressively, in order to obtain all the information related to an assessment.

An architecture overview of the assessment framework is shown in Figure 5.1 where it is possible to differentiate the three tasks previously mentioned. A more detailed description of these concepts is given below. Note that the architecture design allows new capabilities to be dynamically added as plugins into the corresponding levels of the implemented tool.

5.1 Discovery Actions

P0f and Nmap tools are found on the lower levels of our discovery tools selection, where they identify different operating systems or applications which are running on specific devices. Nmap, as an active scanner, provides more output, but it is easily detected by IDS. On the other hand, p0f is hard to be discovered and is also able to detect signatures of devices behind a firewall or NATs. Meanwhile, the results offered by passive discovery are limited or incomplete as O. Arkin describes in [19]. On top of them, a set of tools (described below) are found to fill the different layers of the information model. The Cisco Discovery Protocol [2] (CDP) monitor allowed us to collect even more information and helped to build a network topology. This protocol is

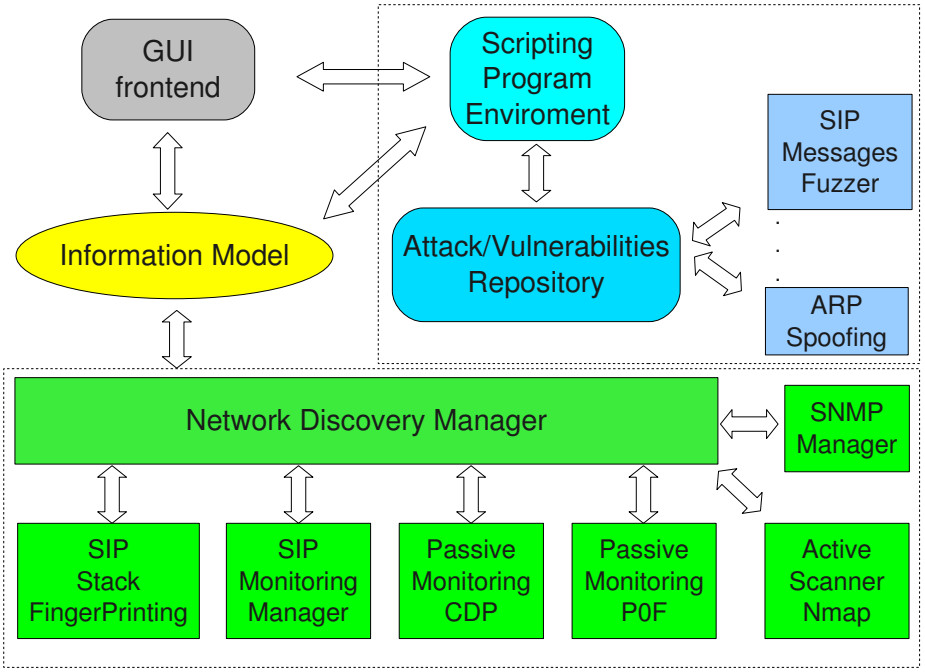


Figure 5.1: Security assessment framework architecture overview

proprietary, but provides the most valuable source of device level information. Concerning the SIP protocol, two fingerprinting detectors specific to this protocol complement the information gathered by the other tools. Finally, the SIP monitoring manager is able to detect which is the current state of a SIP device. All different pieces of information from the different tools are merged together in order to provide a wider view of the current network.

SNMP Manager

A few VoIP devices support SNMP and if the latter is not properly secured, important information can be leaked out by just interrogating the SNMP agent on the device. This entity is capable to do basic SNMP brute forcing and retrieve management related data to be fed into the assessment module.

Cisco Discovery Protocol (CDP) Manager

The Cisco Discovery Protocol [2] is a proprietary protocol used by Cisco devices to advertise themselves and discover other devices in the network. The objective of this protocol is to broadcast the physical configuration of a device to its connected neighbors, thus being useful for the determination of the network topology. The packets generated by the entities supporting this protocol are distributed through multicast and contain relevant information. In the case of SIP phones, a CDP packet exposes a range of data: the actual device model, the firmware version, the device ID (which represents the name of the configuration file if this one is retrieved from a TFTP server), the MAC and IP addresses, the VLAN Domain, and some other less important information. In the functional context, every device periodically sends CDP messages to the multicast address. Cisco entities which support this functionality store the information received to be used as needed. Tools like Yersinia[38] and IRPAS[39] are instantiated in this module because

[38] `http://www.yersinia.net/` last checked on December 2008
[39] `http://www.phenoelit.de/irpas/` last checked on December 2008

of their ability to monitor CDP packets and launch different attacks.

SIP Stack Fingerprinting

Similar to p0f and Nmap, which use signatures to discover the operating system and running services, additional VoIP specific fingerprinting is possible. One possible solution is the one presented by Hong Yan et al. [145] which uses passive and active scanner techniques to find out properties about the SIP entities. The above mentioned work provided a table describing vendor and device specific particularities in their respective SIP stacks. This allows to identify different implementations and to classify them accordingly. For the active discovery part, the idea is to send "OPTIONS" messages and to check the returned set of capabilities. Depending on the answer to malformed messages, the order in the fields and/or the order in the capabilities this method is able to identify some fingerprints. Although a limited scope of devices can be fingerprinted now, an updated database of fingerprint signatures might be a very accurate method.

SIP Syntax Fingerprinting

A second approach for fingerprinting SIP traffic is proposed in this manuscript (Chapter 6). Different with respect to the SIP stack fingerprinting, this approach is passive and does not fingerprint the stack but the message syntax. It is an automated learning process for which the database of signatures can increase more easily if traces from the device are properly found. Its signatures are based on the structure of the message rather than the lexicon as it is explained in the following chapter.

SIP Monitoring Manager

This module is in charge of classifying all the information related to SIP negotiation. Using intercepted packets, it detects the current state of a phone (David Lee et al. in [90] demonstrated it using the OSPF protocol, but the same approach can be achieved for the SIP protocol). From isolated packets carrying information like the out-bound proxy, the SIP entities which the packet has to traverse, etc. can also be recovered.

5.2 Information Model

The main objective of our Information Model is to represent in a structured fashion all the gathered information and to simplify the access to it for any possible assessment. It represents the topology of a network, provided services, applications and any relevant information that can lead to find vulnerabilities.

Standards like the Common Information Model (CIM) [4] describe a sound and commonly used way to design an Information Model. Concretely, it defines a conceptual schema, where it specifies how elements are represented. It is based on UML from which it leverages the object oriented modelling.

The gathered information is obtained in different ways and previous sections in this chapter showed how discovery actions lead to such data. The design objectives must allow for an enhanced usability and capability to represent security related information as well as serve for future reuse. The main module contains a collection for every entity (i.e. devices, services or types of configuration) such that checking for one device in particular should be easily done. The navigation in the model should also allow starting from an entity and getting all the others

pieces of information related to it. A simplified UML diagram of the model is shown in Figure 5.2.

The main module in the Information Model is the Device class, which contains information of the discovered entities: OS, version, etc. It also includes the settings and the running configuration, which are not necessarily the same. The first is related to the meta-configuration, i.e. the values defined to configure the device, while the second are the values used by the device. To illustrate such a case, we define the settings of the IP addresses to be either statically or dynamically configured while the actual IP address of the device belongs to the actual configuration.

Each Device class is associated with a list of Service classes which represent the services hosted by it. The Service entities that are primarily concerned in this model are the ones related to SIP entities:

DHCP Service: if present in a network, some entities depend on it to obtain their IP address. Also, extra information can be obtained, as for instance the routing gateway IP, the DNS server IP, a TFTP server IP, etc. If this service is compromised (i.e. fake IPs information is given by the service), the attacker can set up its own service and do malicious action as explained below.

Routing Gateway: if this service is compromised, the attacker can act as a man in the middle and filter, modify or intercept the VoIP traffic.

DNS Service: entities rely on it to resolve IP addresses. Once the service is compromised, the attacker can resolve URL names to an IP address which is not the real one and for which he/she is providing a rogue service.

TFTP Service: many SIP phones request servers configuration information (i.e. files), that are used for configuration purposes, from TFTP. If this service is compromised, the devices relying on it will be under the control of the attacker.

NAT Service: in a SIP environment it presents an important role because its existence will modify the payload transported at the application level. In order to fake information and to be as convincing as possible, this information has to be well known.

Special interest is focused if the entity provides a SIP service. All SIP traffic is collected for each discovered SIP entity. The traffic is saved following the conceptual structure of SIP Dialogs and Transactions (described in section 1.1.4). Indeed, each packet is decomposed in the fields corresponding to underlying transport layer, SIP (including the body message, e.g. SDP) and RTP messages. The SIP configuration is as well saved following the architectural components of SIP (described in section 1.1.1). For instance, which ports the entity uses for SIP or RTP traffic, which user name, Registrars or Proxies are configured, etc.

Concerning the Information Model, the data can be acquired by two different interfaces which provide more suitability for different tests and attacks.

Classical type: this is the classical syntax of Object Oriented programming languages to navigate through the attributes of classes.

Filtering type: this technique shows an approach focused on the use of filters. Each entity in a class has a function associated to it with the same name as the attribute. This function takes as argument the parameter "filter", which consists of a string that can be validated in each of the current entities. Only the entities satisfying the filter value will be returned in a collection instance. Such a collection provides the same methods and the same attributes

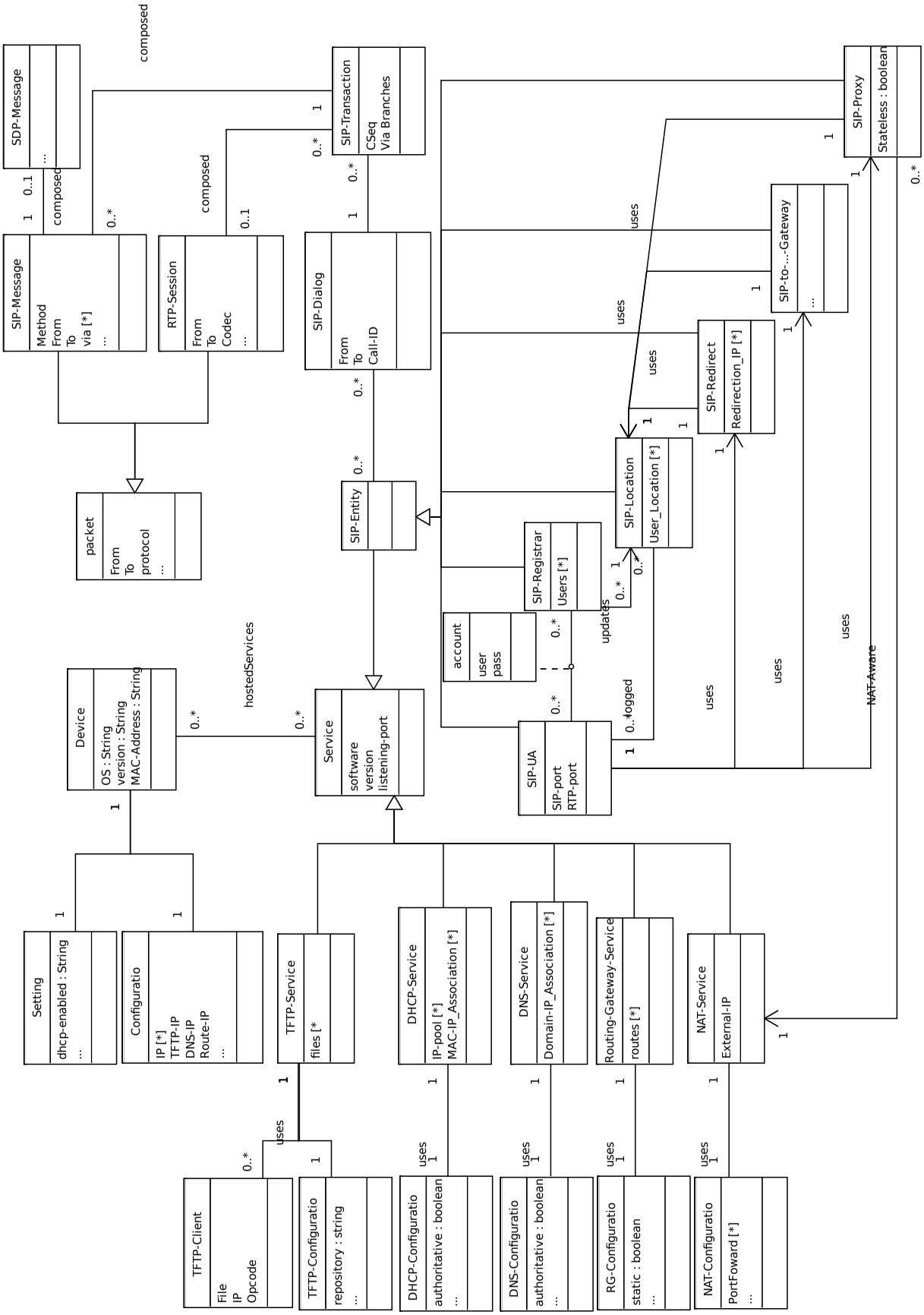


Figure 5.2: Information model UML diagram

declared by the items contained in it and its functionality is to map the methods and attributes to every one of its items.

To illustrate the basic concepts of both syntaxes, consider an example where one desires to retrieve all the TFTP service instances from the devices in the network (i.e. the ones identified by the discovery tools), which contain the specific file "SIP000B46D9CB86.cnf" in their repository. Supposing that the *Model* variable is an instance of the Information Model, the two possible ways to satisfy that request are shown below.

Classical type:

```
tftplist = []
for device in Model.Device:
   for service in device.hostedServices:
      if isinstance(service,TFTP-Service):
         if "SIP000B46D9CB86.cnf" in service.files:
                tftplist.append(service)
return tftplist
```

Filtering type:

```
Model.Device().hostedServices(
   filter="""isinstance(self,TFTP-Service) and
             'SIP000B46D9CB86.cnf' in self.files""")
```

It is worth noting that those two techniques can be freely interlaced.

5.3 Writing Assessment Tests

Once some necessary information was obtained, different kinds of scripts are launched. Those scripts basically exploit some known vulnerabilities to show the degree of exposure of the system. Also, other scripts retrieve new information that is added to the Information Model.

In order to launch different attacks, the framework provides a repository of scripts for which the Information Model is used to represent the necessary data. Among the scripts, attacks like ARP poisoning, Spanning tree attack, DoS, etc. can be launched derived from tools like Yersinia and IRPAS. Other SIP related scripts are SIP-Registrar user enumeration, eavesdropping and RTP play-out, which are instantiated in the fuzzy packet tool [8].

The Attack/Vulnerability repository also allows to store new scripts by simple addition of attacks files. On the fly dynamic testing is possible by our Scripting Program environment which offers an interface to create, try and modify those scripts.

To design an assessment test in order to achieve a complete assessment, multiple conceptual solutions exist, among which the more important are Attack Trees, Attack Graphs, Petri's Network variants, etc. [47, 122, 126]. Our choice for the Attacks Trees design was made for the sake of simplicity, in order to illustrate possible attacks. Once the achievement of a possible attack explained, we proceed to show how it can be accomplished in our scripting environment.

Attack trees are considered to be limited in their functionality, but they provide a good representation of the possible assessment test. In the following an attack tree will be exemplified and we will explain how to perform it using the Information Model described in section 5.2 and the Scripting Environment described in section 5.3.2.

5.3.1 VoIP Attack Tree Example

Figure 5.3 shows an attack tree example where the main objective of this is intercepting a call (extra boxes that are used for the ease of the presentation).

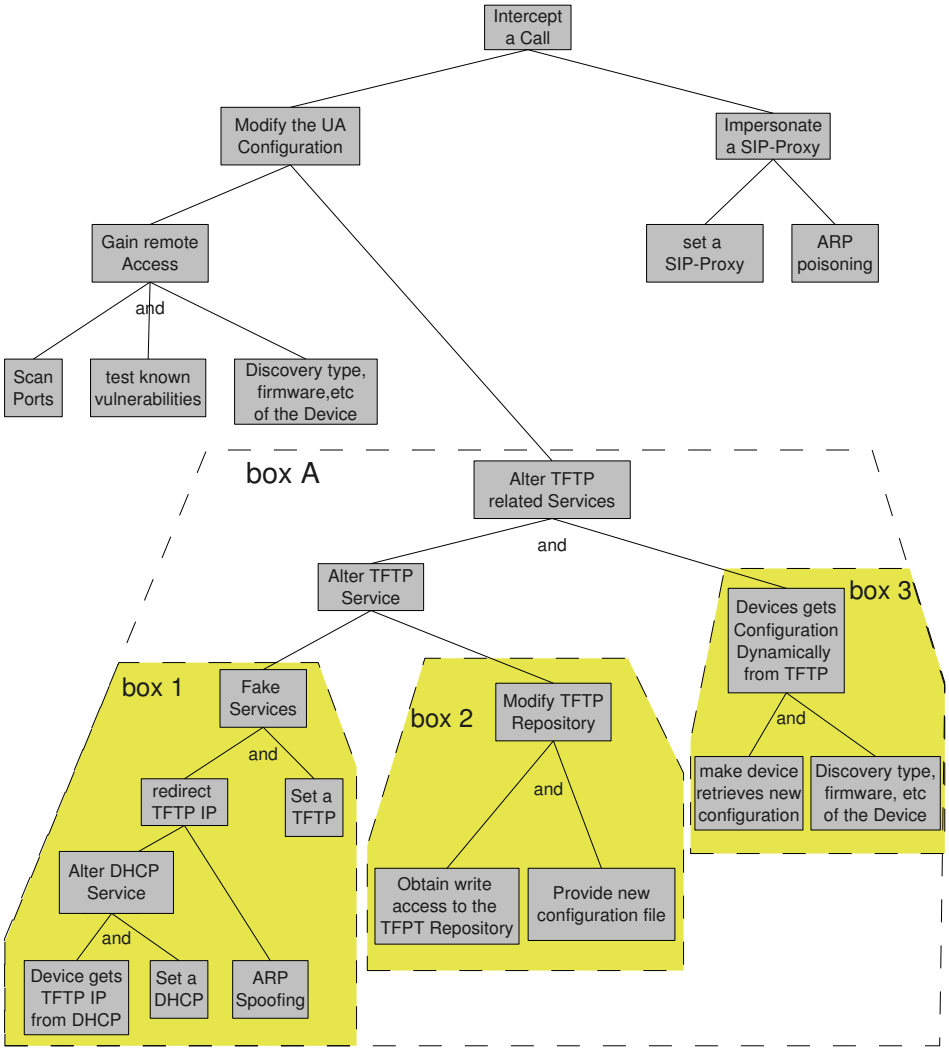


Figure 5.3: Attack tree example

This example shows two possible main branches to achieve this goal.

1. One approach in which the end-devices will not have to be exposed would be to intercept all the out-going SIP transit traffic. This can be possible if, for example, the out-going proxy of the network is impersonated. The next example describes this attack that requires two actions to be done together: 1) Set up a rogue SIP-Proxy in order to provide the needed services and 2) alter the default route to reach the original proxy via the rogue one. There are several ways to achieve the latter action, like for instance ARP poisoning, Spanning Tree attacks, DNS poisoning.

2. The second approach aims at changing the configuration of the SIP User Agent. It also shows some possible ways to accomplish it.

   (a) The first goal is to gain remote access to the device (if possible), which can be done if the software is not up to date, for example. The steps described next aim at discovering as much information as possible, and the use of tools like Nmap help

to discover the open ports on the device as well as running services on each of them (including the version and software). Once all that information is processed, databases of known vulnerabilities as US-CERT[40], can be searched for the respective software and version, and one can then check for known vulnerabilities and exploit them.

(b) Secondly, the assessment tester can alter the services which a SIP UA is relying on (e.g. DHCP server, TFTP server, etc.). Thus, those services can be compromised so that every UA retrieving its configuration from the TFTP server gets rogue information, which allows the attacker to intercept all its SIP traffic. This step, shown in the Figure 5.3, is framed in the *Box A*. Each of its sub-boxes (i.e. 1, 2 and 3) regroups related activities. Note that Box 3 has to be done in conjunction with Box 1 or 2 and only afterwards, not as standalone:

- **Box 1:** This attack shows how to redirect all traffic to a rogue service. The first thing that the attacker has to do is to set up a TFTP server. Once done, all the traffic directed to the target TFTP has to be transferred to the machine running this rogue TFTP service. One possible action to achieve this redirection is to poison the network with spoofed ARP packets. This poisoning has to be done just to the target devices or in switches close to the service. Other techniques exist to accomplish the same, as, for example, Spanning Tree attacks, but are out of the scope of this manuscript. Another solution to redirect the traffic is to directly set up the IP address of the fake TFTP server on the target device. This can be easily done if the device is getting its TFTP IP address from the DHCP service. If that is the case, the attacker can run such a service with this fake information and overwrite the genuine data.
- **Box 2:** This approach shows a way where the files to be retrieved from the TFTP server are overwritten. First the attacker has to know which files he wants to overwrite or create. The identification can be done by a passive scanner which just reads traffic on the network and interprets messages under the TFTP protocol. This is possible, since no encryption is usually used in this step. Additionally, some devices use a file name that is built using well know prefixes and postfix attached to the MAC address of the device. Once the files are known, the TFTP server can be tested to see if write access is granted on it and if possible, these files are modified with rogue configuration data.
- **Box 3:** Finally the objective of this sub-tree framed in the box is to identify which are the devices that retrieve their configuration from the TFTP servers and make them reload the new configuration. The identification can be done as explained in the Box 2 by a passive scanner which is listening to the TFTP protocol. Once the TFTP dependent devices are identified, information concerning their OS, software, version, etc. will help to find vulnerabilities that will force the device to retrieve the new configuration (e.g. reboot the device).

It is worth noting that those attacks are very naive and if they accomplish the objective, it is very probable that they will be detected by an IDS. The contribution of this example is to show how automation of these tasks can be done.

5.3.2 Scripting Environment

This extension represents the active assessment part of the architecture. It provides an environment where pre-defined tests can be retrieved from a repository and where new ones can be

[40] `http://www.kb.cert.org/vuls/` last checked on December 2008

generated on the fly, with the possibility of storing the procedure in the repository for later use.

One of such scripts is the SIP Messages Fuzzer. Security testers of protocol implementations have shown that malformed message fields are able to crash or exploit some entities. In a well known case, only one out of nine SIP phones was able to pass the tests performed by the "PROTOS Test-Suite: c07-sip" [135]. One single test may contain one or more exceptional elements which can range from exceptional IPv4 addresses to malformed tags. The tests are very simple and mostly probe the robustness of the parser, but they show the weakness in most implementations. Further on, chapter 8 focuses in the searching of vulnerabilities using fuzzing techniques and described one of the main contributions of this manuscript.

As the core implementation and the wrappers (i.e. wrappers for the external tools used) are coded in Python, all tests are specified as a Python script and the interactive environment is a derivation of the one from Python.

Python Language Embedding

Lately, tools like scapy[41] (a Packet Manipulator Program) are gaining popularity, mostly because their usability and flexibility of programming. Scapy allows the user to read and inject packets, with all the desired custom options, in an extremely easy manner. Our approach was designed with the same goals in mind, to include functionalities developed by these tools and complement them with information from application levels. Thus, the information included is the one described in the Information Model which will provide all the data obtained by the discovery actions. Meanwhile, the interactive environment corresponds to an extension of the Python interactive environment which provides functionalities such as a low level packets generator and pre-defined high level attacks.

Example Script

Figure 5.3 in box 1, shows a sub-attack that, as explained previously, tries to redirect the real TFTP server IP to a rogue one.

To alter the DHCP Service, first a rogue server must be running in a vulnerable machine, which will try to distribute the IP of the fake TFTP server.

To create an ARP spoofing attack the repository provides a predefined script, which may require some information like the target IP, the IP of the device that has to be impersonated and the MAC to redirect the packets. For such attacks, the application has to be launched inside the LAN. The script attack is illustrated in Figure 5.4.

The second alternative to alter a TFTP Service described as Box 2 (Figure 5.3), i.e. for discovery if writing access is granted, should proceed as in Figure 5.5.

Finally, the third box shown in Figure 5.3, which should work in conjunction with the box 1 or 2, requires that the device requests the new rogue configuration from the TFTP server. Some SIP hard phones reboot after specific malformed messages or simpler they could reload the dial plan by sending a NOTIFY message with the option event "check-cfg". The latter depends on the current configuration. Figure 5.6 shows the last required script for the whole test.

5.4 Conclusion

Our current research, addresses the secure management of VoIP networks. One of our main work direction is an integrated system able to retrieve as much as possible the information of

[41] `http://www.secdev.org/projects/scapy/` last checked on December 2008

```
###  Retrieves all the Service instances of the devices                ###
###  which provides TFTP                                               ###
>>> TFTPServices = Model.Services(filter=
                                        "isinstance(self,TFTP_Service)")

###  Start ARP spoofing for each Server                                ###
>>> for tftp_server in TFTPServices:
...     tftpIP = tftp_server.Device.Configuration.IP

###      For each of the IP of the clients of this TFTP               ###
...     for clientIP in tftp_server.TFTP_Client().IP:

###              Launch ARP poisoning attack                          ###
>>>             Attack.ARP_Poisoning(
                                    targetIP= clientIP,
                                    impersonateIP= tftpIP,
                                    fakeMAC= fakeMAC
                                   )
```

Figure 5.4: ARP poisoning script

```
>>> from scapy import sr1,IP,UDP
>>> from scapy.extras import TFTP
>>> grantedAccess = []

###  Retrieves all the Service instances of the devices                ###
###  which provides TFTP                                               ###
>>> TFTPServices = Model.Services(filter=
                                        "isinstance(self,TFTP_Service)")

###  Check writing access for each Server                              ###
>>> for tftp_server in TFTPServices:
...     tftpIP = tftp_server.Device.Configuration.IP
...     for file in tftp_server.files:

###      Use scapy to inject TFTP packets                             ###
...     p = sr1(
                IP(dst=tftpIP)/
                UDP()/
                TFPT(dst_file=file,Opcode=WRITE_REQUEST)
               )

###      Check the answer for writing access                          ###
...     if p[3].Opcode != ERROR_CODE:
...             grantedAccess.append((tftpIP,file))
```

Figure 5.5: TFTP service alteration

the environment. Such data can be used by assessment tests and can show scenarios allowing to identify where the flaws of the system are located.

In this chapter we mainly focused on VoIP networks but we consider that this approach as well as the architecture model can be extended to several other types of networks. We show how accurate information from the network can be gathered, and next provide an Information Model

```
>>> from scapy import sr1,IP,UDP
>>> from scapy.extras import SIP

###  Retrieves all the UAs with the version that could be rebooted   ###
>>> sipUAs = Model.Services(
                              filter="isinstance(self,SIP-UA)
                              and self.software = 'XXXX'
                              and self.version <= 'X.XX'"
                            )

###  Use scapy to inject the NOTIFY packets                           ###

>>> for UA in sipUAs :
...     UA_IP = UA.Device.Configuration.IP
...     p = sr1(
                IP(dst=UA_IP)/
                UDP()/
                SIP(
                    Method="NOTIFY",
                    Event="check-cfg"
                   )
              )
```

Figure 5.6: Reload dial plan using the NOTIFY message

capable to represent it in an appropriate way for assessment methods.

Three main contributions have been addressed: we propose a network information model capable to represent the information required to perform VoIP assessment, we describe a VoIP assessment architecture and its implementation and we build a framework based on attack tree modelling in order to represent and write VoIP attacks.

Part III

Advanced Structural Fingerprinting

Chapter 6

Fingerprinting

Security assessment tasks and intrusion detection systems do rely on automated fingerprinting of devices and services. Most current fingerprinting approaches use a signature matching scheme, where a set of signatures is compared with traffic issued by an unknown entity. The entity is identified by finding the closest match with the stored signatures. These fingerprinting signatures are found mostly manually, requiring a laborious activity and needing advanced domain specific expertise. This chapter describes a novel approach to automate this process and build flexible and efficient fingerprinting systems which deviated in the following publication [11]. This system it is able to identify the source entity of messages in the network. A passive approach is followed without need to interact with the tested device. Application level traffic is captured passively and inherent structural features are used for the classification process. A new technique is described and assessed for the automated extraction of protocol fingerprints based on arborescent features extracted from the underlying grammar.

Most known application level and network protocols use a syntax specification based on formal grammars. The essential issue is that each individual message can be represented by a tree like structure. We have observed that stack implementers can be tracked by some specific subtrees and/or collection of subtrees appearing in the parse trees. The key idea is that structural differences between two devices can be detected by comparing the underlying parse trees generated for several messages. A structural signature is given by features that are extracted from these tree structures. Such distinctive features are called fingerprints. We will address in the following the automated identification of them.

If we focus for the moment on individual productions (in a grammar rule), the types of signatures might be given by:

- Different **contents** for one field. This is in fact a sequence of characters which can determine a signature. (e.g. a prompt or an initialization message).

- Different **lengths** for one field. The grammar allows to produce a repetition of items (e.g. quantity of spaces after a symbol, capabilities supported). In this case, the length of the field is a good signature candidate.

- Different **orders** in one field. This is possible, when no explicit order is specified in a set of items. A typical case is how capabilities are advertised in current protocols.

We propose a discrimination method able to automatically identify distinctive structural signatures. This is done by analyzing and comparing captured messages traces from the different devices.

6.1 Grammar Inference

6.1.1 ABNF Grammars

The key assumption made in our approach is that an Augmented Backus-Naur Form [41] (ABNF) grammar specification is a priori known for a given protocol. Such a specification is made of some basic elements as shown in Figure 6.1.

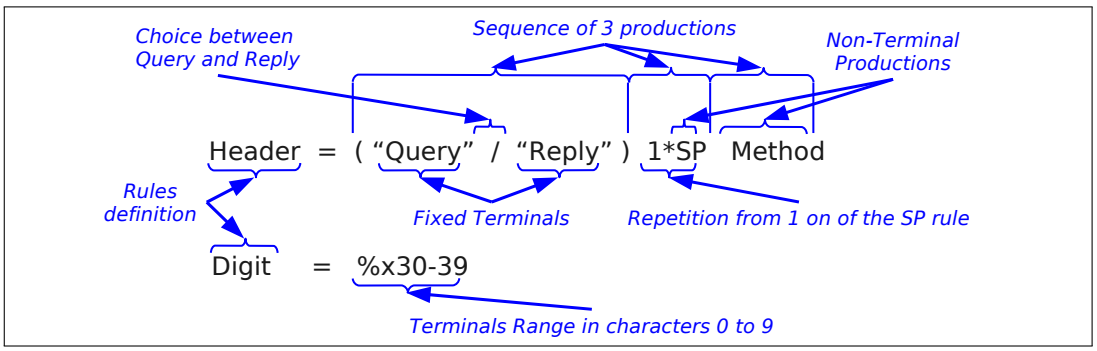


Figure 6.1: Basic elements of a grammar

- A **Terminal** can represent a fixed string or a character to be chosen from a range of legitimate characters.
- A **Non-Terminal** is reduced using some rules to either a Terminal or a Non-Terminal.
- A **Choice** defines an arbitrary selection among several items.
- A **Sequence** involves a fixed number of items, where the order is specified.
- A **Repetition** involves a sequence of one item/group of items, where some additional constraints might be specified.

Formally a grammar of the type consists of 4-tuple $G = (\Sigma, N, P, n_0)$ where:

$\Sigma =$ finite set of terminals (string literals).

$N =$ finite set of non-Terminals.

$P =$ finite set of mapping rules of the form $P : N \rightarrow e$, where e is an *expression* as described bellow.

$n_0 =$ a non-Terminal called the starting symbol.

An inductive definition of the *expressions*, where it is assumed that e, e_1, .., e_n are *expressions* as well, is as follows:

- Terminals.
- non-Terminals.
- Sequences e_1 .. e_n
- Choices e_1 / .. / e_n

- k-Repetitions $e^{(i,j)}$ where $0 \leq i \leq k \leq j$

Note that some assumptions were made for the sake of simplicity. All the other expressions not mentioned (Character Classes Terminals, Incremental Choices, Sequence Groups, Specific Repetitions and Optional Sequences) can be simulated by the previous expressions.

6.1.2 Structural Inference

A given message is parsed according to the fields defined in the grammar. Each element of the grammar is placed in a n-ary tree which obeys to the following rules:

- A **Terminal** becomes a leaf node with a name associated (i.e. the terminal that it represents) which is associated to the encountered value in the message. Indeed, the established order of the leaf from left-to-right is the message content itself.
- A **Non-Terminal** is an internal node associated to a name (i.e. the non-terminal rule) and it has an unique child which can be any of the types defined here (e.g. Terminal, non-Terminal, Sequence or Repetition).
- A **Sequence** is an internal node that has a fixed number of children. This number is in-line with the rules of the syntax specification.
- A **Repetition** is also an internal node, but having a number of children that may vary based on the number of items which appear in the message.
- A **Choice** does not create any node in the tree. However, it just marks the node that has been elected from a choice item.

It is important to note that even if sequences and repetitions do not have a defined name in the grammar rules, an implicit name is assigned to them that uniquely distinguishes each instance of these items at the current rule.

Figure 6.2 shows a Toy ABNF grammar defined in (a), messages from different implementation compliant with the grammar in (b/c) and (d) the inferred structure representing one of the messages in (d).

6.1.3 Grammar Paths

Grammar paths have been defined to provide the ability to navigate around the tree representation of the message, selecting nodes by a variety of criteria. Indeed, it may define a relative or exact position. Besides, they will become quite useful in the tree comparison process as explained in section 6.2.1.

A grammar path is basically a sequence of tag nodes starting from a specific node (usually the root node unless otherwise stated) finishing in another node which must be a descendant of the initial node.

Each node tag in the tree representation of the message is represented in a grammar path following different constraints depending on its node type:

Terminal items are represented in the grammar path with the regular expression or fixed string that they defined.

non-Terminal items are represented by the rule name

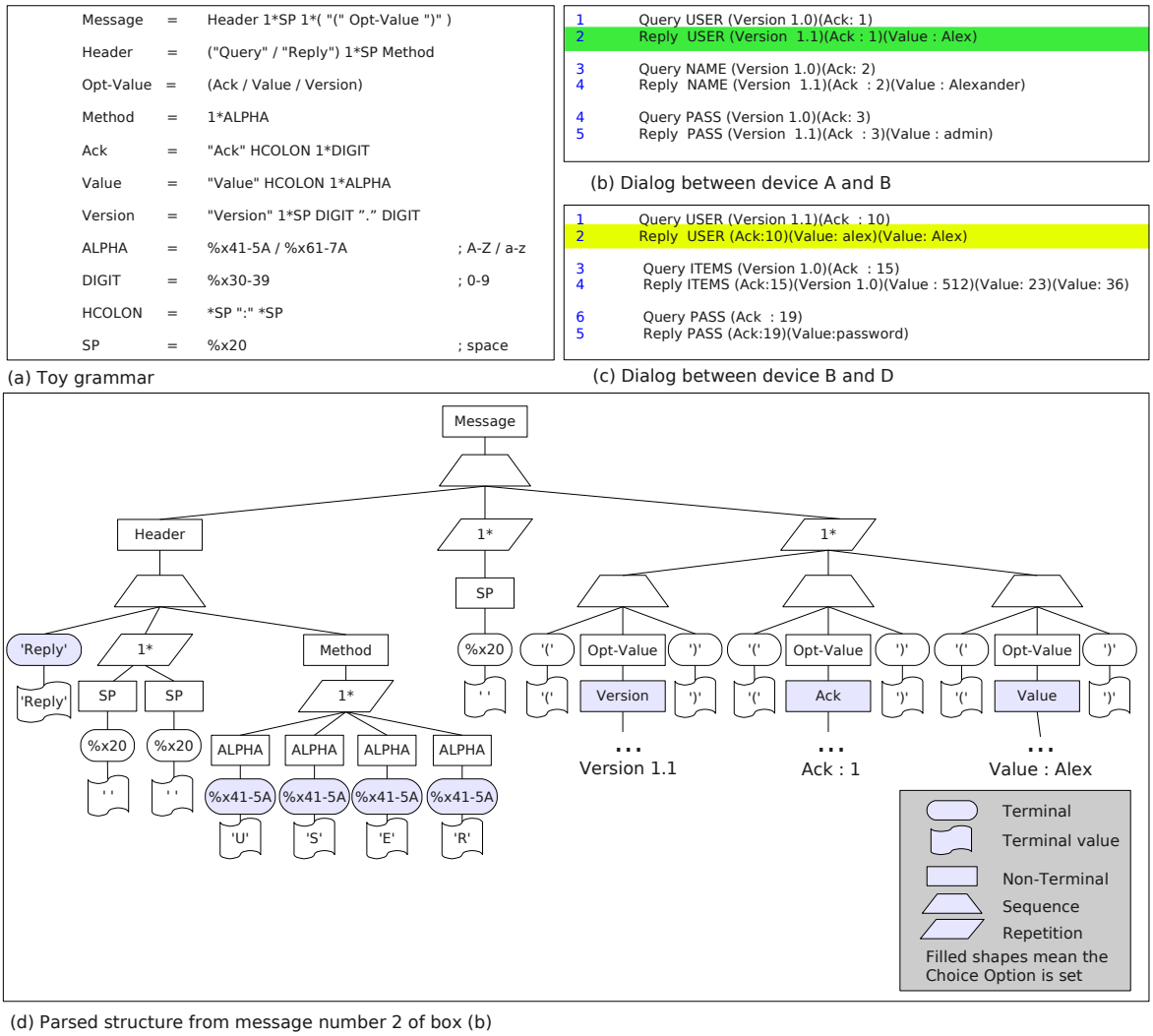


Figure 6.2: Parsed structure grammar

Choice items can be represented by the name of the chosen rule or by its appearance order in the choice selections (starting from 0).

Sequence items are represented by the number that represent its order position in the sequence (starting from 0). Note that the sequence itself is not specified in the path.

Repetition items are represented as **(i)**, where i is the order position as how it appears in the message. Note that the repetition itself is not specific in the path as well.

It is worth noting that the use of wildcards are allowed, either quote(?) or star(*). Quotes are wildcards useful to just match all the children from the current node. Note that if the current step refers to a repetition, then it has to be encapsulated by parenthesis as if it was a position. While the star represents a wildcard which will match all the nodes in the current subtree.

Figure 6.3 illustrates several possible grammar paths linked to one message representation.

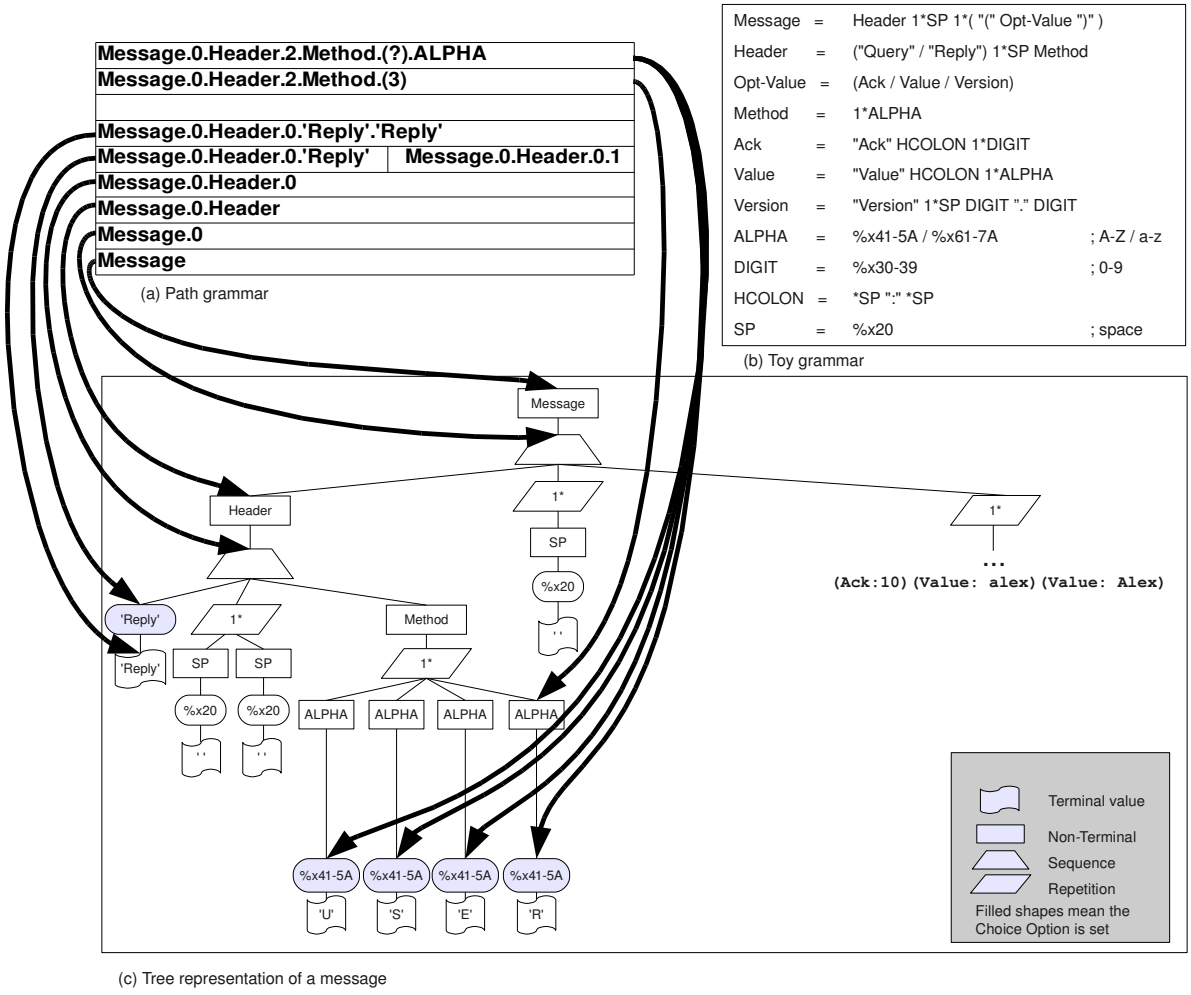


Figure 6.3: Grammar paths

6.2 Node Comparison

One of the challenges posed by this work was based on the automation of the signature discovery, therefore an effective function for comparing messages is the corner stone of the approach. Assuming that we have two SIP-URIs (i.e. SIP addresses) with a structure as shown in Figure 6.4, then it will be incorrect to compare a *User Name* with a *URL* even if they have syntactically much in common. In fact, their lexicon is similar but their semantic is totally different.

Then, the underlying grammar of the protocol syntax becomes an interesting candidate to automatically identify the semantic of the lexicon in a message. Therefore, the comparison of messages resemble a comparison of trees as detailed in the following sections.

6.2.1 Node Signatures and Resemblance

Comparison between nodes is summarizes to two steps: 1) find an appropriate object that represents the attributes of each of the nodes, called a *Node Signature* and 2) the resemblance function is computed from the *Node Signatures* identified from the two nodes.

While comparing two nodes we expect that the comparison takes places over two semantically equivalent nodes. It is worth noting that nodes inside a repetition may be equivalent regardless

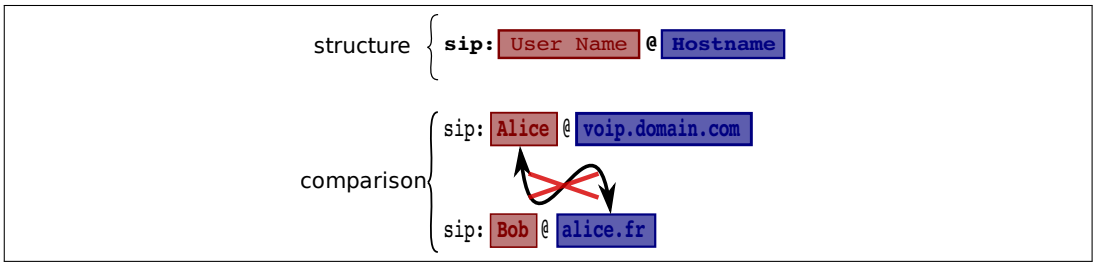


Figure 6.4: Comparison challenges

of their order on it. Therefore, the guidelines for designing the *Node Signature* (for a tree or a sub-tree) should follow some principles like:

1. As more items are shared between trees, more similar must be their signatures.

2. Nodes that have different tags or ancestors must be considered different.

3. In cases where the parent node is a sequence, the location order in the Sequence should be represented somehow in the tree signature.

4. If the parent node is a repetition, the location order should not be part of the tree signature, order will be captured later on the fingerprinting features.

Different approaches exist in the literature targeting different goals. P. Zezula et al. [146] define a tree signature as an n-tuple where n is the number of nodes in the tree. Each item represents a node in the tree and their order in the n-tuple depends in the pre-order of appearance in the tree. The representation of each node is specified as a 2-tuple containing the label and the post-order of the node in the tree. As a consequence the information from the signature, like the node depths, cardinality, etc. can be easily acquired. More important checking whether subtrees are included in other trees is straightforward. However, this approach is not as flexible as we require, since the order of the repetitions is preserved. Thus, our fourth principle is not satisfied.

S. Flesca et al. [62] present a structural comparison of XML documents were they encode the document with a bijective function that creates a sequence of values. Those values depend in a tag encoding and in a document encoding, for which they propose a multilevel encoding able to give different results according to the node name plus the ancestor value. Once the encoding is done, suppose $seq =< a_0, .., a_n >$, they define a function $f(i) = a_i$ where $0 \leq i \leq n$. This function normalized is given as an input to a Discrete Fourier Transformation (DFT), which conserves some good properties regarding the tree structure. Then, the comparison between the DFT transformations defines the similarity of the trees. However, this comparison is rather non intuitive and will match two nodes with similar subtrees even so they are semantically different. Thus, this approach fails to satisfy the second principle, as it focuses on the matching based on the subtrees with similar structures ignoring different ancestor tags.

Finally, the approach published by D. Buttler in [36] is the one closest to satisfy all the principles. This method starts by encoding the tree in a set. Each element in the set represents a partial path from the root to any of the nodes in the tree. A resemblance method defined by A. Broder [34] uses the elements of the set as tokens. This resemblance is based on shingles, where a shingle is a contiguous sequence of tokens from the document. Between documents D_i

Partial Paths	Occurrences
Header.0.'Reply'	1
~~Header.0.'Reply'.'Reply'~~	1
Header.1.?	2
~~Header.1.?.SP~~	2
~~Header.1.?.SP.%x20~~	2
~~Header.1.?.SP.%x20.' '~~	2
Header.2.Method.?	4
~~Header.2.Method.?.ALPHA.~~	4
Header.2.Method.?.ALPHA.%x41-5A	4
Header.2.Method.?.ALPHA.%x41-5A.'U'	1
Header.2.Method.?.ALPHA.%x41-5A.'S'	1
Header.2.Method.?.ALPHA.%x41-5A.'E'	1
Header.2.Method.?.ALPHA.%x41-5A.'R'	1

(~~strikethrough~~) Strikethrough paths are the ones considered as cosmetics.
(?) Quotes define that the current path may be any of the repetition items.

Table 6.1: Partial paths obtained from figure 6.2 (d)

and D_j the resemblance is defined as:

$$r(D_i, D_j) = \frac{|S(D_i, w) \bigcap S(D_j, w)|}{|S(D_i, w) \bigcup S(D_j, w)|} \tag{6.1}$$

where $S(D_i, w)$ creates the shingles of length w for the document D_i.

Definition 1. *The **Node Signature** function is defined to be a Multi-Set of all partial paths belonging to the sub-branch of the node.*

The *partial paths* start from the current node rather than from the root of the tree, but still go through all the nodes of the subtree which has the current element as its root like it was in the original approach of D. Buttler. However, partial paths obtained from fields classified as *Cosmetics* are excluded from this Multi-Set. The structure used is a Multi-Set rather than a Set in order to store the quantity of occurrences for specific nodes in the sub-branch. For instance, to illustrate the difference, the number of spaces after a specific field can determine a fingerprint signature in an implementation.

It must be considered that sibling nodes in a sequence item are fixed and representative. However, sibling nodes in a repetition are not representative. This repetition items are thus represented in the *partial paths* of the Multi-Set using a quote wildcard rather than using their respective position in the repetition.

Table 6.2.1 shows the *Node Signature* obtained from the node *Header* in the tree of Figure 6.2 (d).

Definition 2. *The **Resemblance** function used to measure the degree of similarity between two nodes is based on the Equation 6.1. The $S(N_i, w)$ function applies the Node Signature function over the node N_i.*

Using $w = 1$ allows to compare the number of items these nodes have in common though ignoring their position with respect to their siblings for the case of repetitions.

6.2.2 Structural Difference Identification

Algorithm 1 is used to identify differences between two nodes which share the same ancestor path in the two trees, where the functions *Tag*, *Value*, *Type* return the name, value and respectively

Algorithm 1 Node differences Location

```
procedure NODEDIFF(node_a, node_b)
    if Tag(node_a) = Tag(node_b) then
        if Type(node_a) = TERMINAL then
            if Value(node_a) != Value(node_b) then
                Report_Difference('Value', node_a, node_b)
            end if
        else if Type(node_a) = NON-TERMINAL then
            NODEDIFF(node_a.child_0, node_b.child_0))        ▷ Non_Terminals have
                                                             ▷ unique child
        else if Type(node_a) = SEQUENCE then
            for i = 1..#node_a do                            ▷ In a Sequence
                NODEDIFF(node_a.child_i, node_b.child_i)     ▷ #node_a = #node_b
            end for
        else if Type(node_a) = REPETITION then
            if not (#node_a = #node_b) then
                Report_Difference('Length', node_a, node_b)
            end if
            matches := Identify_Children_Matches(node_a, node_b)
            if ∃ (i, j) ∈ matches : i != j then
                Report_Difference('Order', node_a, node_b)
            end if
            for all (i, j) ∈ matches do
                NODEDIFF(node_a.child_i, node_b.child_j)
            end for
        end if
    else
        Report_Difference('Choice', node_a, node_b)
    end if
end procedure
```

the type of the current node. Note that the following property applies:

$$Tag(node_a) = Tag(node_b) \Rightarrow Type(node_a) = Type(node_b)$$

The function **Report_Difference** takes the type of difference between the two nodes and reports it. Each time the function is called, it creates one structure that stores the type of difference, the partial path from the root of the tree to the current nodes (which is the same for both nodes) and a corresponding value. For differences of type 'Value' it will store the two terminal values, for 'Choice' the two different Tag names, for 'Length' the two lengths and for 'Order' the matches.

The function **Identify_Children_Matches** identifies a match between children of different repetition nodes. The similitude between each child from $node_a$ and $node_b$ (with n and m children respectively) is represented as a matrix, M, of size n x m where:

$$M_{i,j} = resemblance(node_a.child_i, node_b.child_j)$$

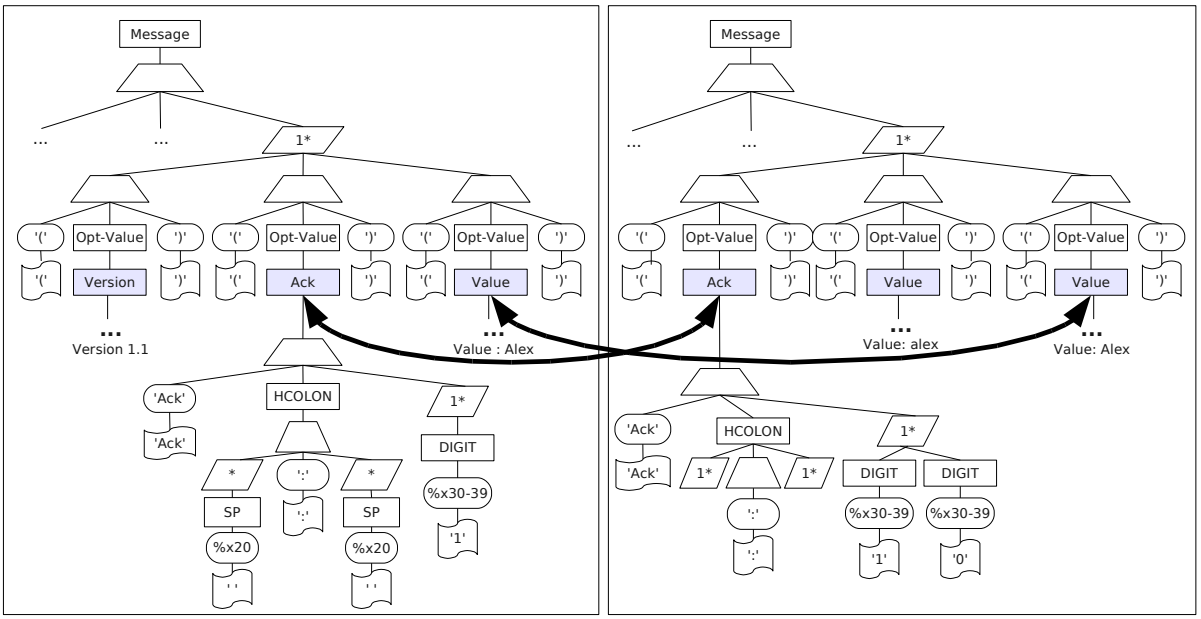
To find the most adequate match, a greedy matching assignment is used. Children with the biggest similarity score are match. If a child from $node_a$ shares the same similarity score with more than one child from $node_b$, then the nodes having a closer distance (respecting to their position in the repetition item) will match.

Figure 6.5 illustrates an example match. Boxes (a) and (b) correspond to the portion of the tree representations been compared. The matching process first compute the Node Signature for

each item on both trees (the repetition items are highlighted in blue). Box (c) shows the obtained node signatures from the Ack nodes (i.e. the partial path, type and item values for each node). It is worth noting that the partial paths list does not include the parenthesis items wrapping each node. Those items are called *Cosmetics* nodes, which are required in the message in order to be compliant with the grammar but they do not provide useful information. On the contrary, they can erroneously match uncommon nodes; since their resemblance will be bigger than zero just because of the delimiters even though they do not share any other similarity. Therefore, this cosmetics nodes are no included in the signature.

The second step of the matching algorithm consists in computing a pairwise resemblance between each repetition items of the two nodes. Box (d) shows the resemblance results for the node signatures obtained from the two Ack nodes. Thus we can obtain the following matrix using the *Resemblance* method with the path "`Message.2.?`". The rows in the matrix represent the children from the subtree in (a) and the columns the children from subtree (b). Therefore, the bold numbers in the matrix represent the performed match based on their highest score.

$$M = \begin{pmatrix} .00 & .00 & .00 \\ \mathbf{.33} & .00 & .00 \\ .00 & .61 & \mathbf{.90} \end{pmatrix}$$



(a) Message number 2 of figure 1 (b)

(b) Message number 2 of figure 1 (c)

Partial Path	Type	$Node_a$	$Node_b$
ACK.1.HCOLON.0	Static Length	1	0
ACK.1.HCOLON.2	Static Length	1	0
ACK.2	Static Length	1	2
ACK.2.(?).DIGIT.%x30-39	Static Value	'1'	'1'
ACK.2.(?).DIGIT.%x30-39	Static Value		'0'

(c) Node signatures from the **Ack** nodes

$\lvert S_A \cup S_B \rvert$	$\lvert S_A \cap S_B \rvert$	$\frac{\lvert S_A \cap S_B \rvert}{\lvert S_A \cup S_B \rvert}$
1	0	0
1	0	0
2	1	0.5
1	1	1
1	0	0
6	2	0.33

(d) Shingle information from the **Ack** nodes where $S_a = S(N_a,1)$ and $S_b = (N_b,1)$

Figure 6.5: Performed match between sub-branches of the tree

6.3 Fingerprinting Automation (Signature Discovery)

The overview of the training and classification process is illustrated in Figure 6.6.

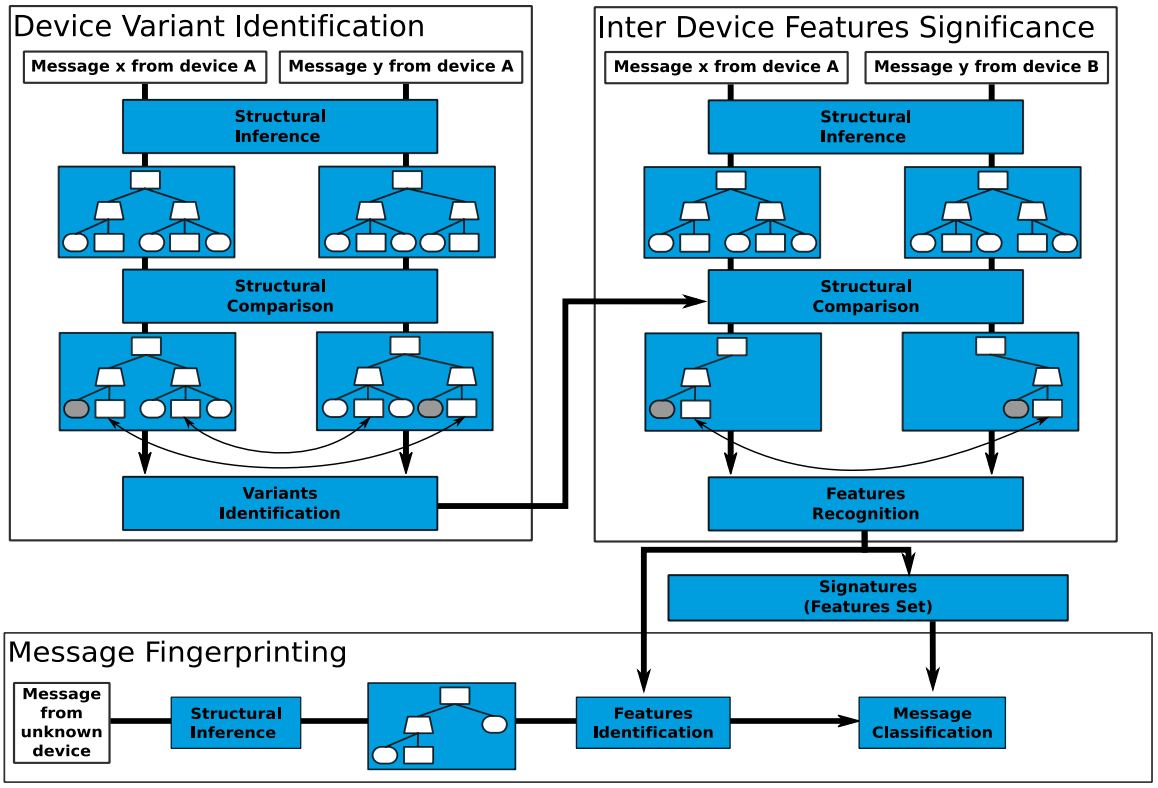


Figure 6.6: Fingerprinting training and classification

The upper boxes in Figure 6.6 constitute the training period of the system. The output is a set of signatures for each device presented in the training set. The lowest box represents the fingerprinting process. The training is divided in two phases:

Phase 1 (*Device Variant Identification*). In this phase, the system automatically classifies each field in the grammar. This classification is needed to identify which fields may change between messages coming from the same device.

Phase 2 (*Inter Device Features Significance*) identifies among the Invariant fields of each implementation, those having different values for at least two group of devices. These fields will constitute part of the signatures set.

When one message has to be classified, the values of each invariant field are extracted and compared to the signature values learned in the training phase.

6.3.1 Phase 1: Variant Identification

One major activity that was not yet described is how non-invariant fields are identified. The process is done by using all the messages coming from one device, comparing them and finding the differences between each two messages using algorithm 1. For each result, Algorithm 2 is executed in order to fine tune the extracted classification. This algorithm recognizes only the

Algorithm 2 Fields Classification Algorithm

```
procedure FieldClassification(differences_{a,b})
    for all diff ∈ differences_{a,b} do
        if diff.type ==' Value') then
            Classify_as_Variant('Value', diff.path)
        else if diff.type ==' Choice' then
            Classify_as_Variant('Choice', diff.path)
        else if diff.type ==' Length' then
            Classify_as_Variant('Length', diff.path)
        else if diff.type ==' Order' then
            if not (∀ (i, j), (x, z) ∈ diff.matches :
                    (i < x ∧ j < z) ∨ (i > x ∧ j > z)) then
                                ▷ Check if a permutation exists between the matched items.
                Classify_as_Variant('Order', diff.path)
            end if
        end if
    end for
end procedure
```

fields that are *Variant*. The set of *Invariant* fields will be represented by the union of all the fields not recognized as *Variant*.

The **Classify_as_Variant** function stores in the list **fieldClassifications**, for each of the differences, a tuple with the partial path and the type of the found difference (e.g. 'Value', 'Choice', 'Length' or 'Order').

Assuming a training set Msg_set, of messages compliant with the grammar as

$$Msg = \bigcup_{i=0}^{n} msg\_set_i$$

where n is the quantity of devices and msg_set_i is the set of messages generated by device i, the total number of comparisons computed in this process is

$$\sum_{i=0}^{n} \frac{|msg\_set_i| * (|msg\_set_i| - 1)}{2}$$

6.3.2 Phase 2: Invariant Identification

In contrast to the Fields Classification, this process compares all the messages from the training set sourced from different devices. All the *Invariant* Fields -for which different implementations have different values- are identified and saved as features. Algorithm 3 describes how these features are recognized. The algorithm inputs are the *fieldClassifications* computed by algorithm 2, the Devices Identifier to which the compared messages belong as well as the set of differences found by Algorithm 1 between the messages.

The **add_Feature** function stores in **recognizedFeatures**, the partial path of the node associated with the type of difference (i.e. Value, Name, Order or Length) and a list of devices with their encountered value. However, the 'Order' type represents a more complex approach, requiring a mapping function between the nodes in the repetition and their position on it.

Concretely, order features are represented as Partially Ordered Sets (POSet). The algorithm requires two stages to achieve this poset. First, all the children nodes for the repetition items (over all the messages) are extracted and their *node signatures* are computed. The node signatures are stored as the items of the poset. Once the minimal set of signature items has been identified, the second phase identify the matches of the repetition children into the existing items in the poset (using the *resemblance* function). The order of the current repetition is then computed based on how the items in the message match the ones from the poset.

Algorithm 3 Features Recognition Algorithm

```
procedure FEATURESRECOGNITION(fieldClassifications, DevID_{a,b},
                              differences_{a,b})
    for all diff ∈ differences_{a,b} do
        if not (diff.type, diff.path) ∈ fieldClassifications then
            if diff.type == 'Value' then
                addFeature('Value', diff.path, DevID_{a,b}, diff.value_{a,b})
            else if diff.type == 'Choice' then
                addFeature('Choice', diff.path, DevID_{a,b}, diff.name_{a,b})
            else if diff.type == 'Length' then
                addFeature('Length', diff.path, DevID_{a,b}, diff.length_{a,b})
            else if diff.type == 'Order' then
                if (∃ (x, z) ∈ diff.matches : x ≠ z) then
                    addFeature('Order', diff.path, DevID_{a,b},
                                        diff.match, diff.children_nodes_{a,b})
                end if
            end if
        end if
    end for
end procedure
```

Assuming the earlier Msg_set set, the invariant identification process will do the following number of comparisons:

$$\sum_{i=0}^{n} |msg\_set_i| * \sum_{j=i+1}^{n} |msg\_set_j|$$

From the *fieldClassifications*, only the *Static* fields are used to fill the *recognizedFeatures*. The recognized features define a sequence of items, where each one represents the field location path in the tree representation and a list of Device ID with their associated value.

The Recognized Features can be classified in:

- Features that were found with each device and for which at least two distinct values are observed for a pair of devices.
- Features that were found in some of the devices for which such a location path does not exist in messages from other implementations.

6.3.3 Fingerprinting

The classification of a message uses the tree structure representation introduced in section 6.1. The set of recognized features obtained in section 6.3.2 represents all the partial paths in a tree structure that are used for the classification process.

In the cases where the features are of type *'Value'*, *'Choice'* or *'Length'*, their corresponding values are easily obtained. However, the case of an *'Order'* represents a more complex approach, requiring some minor improvements.

First, the *Node Signature* from the node's children are computed, as defined in Section 6.2.1. Then, from the set of Order features, a matching is done to find the item in the set that has most resemblance to the current item. Once the matching is obtained, the order of the children is checked against all the existing orders for the feature. The devices where their order respect the message order are then possible candidates for the classification.

Figure 6.7 illustrates some identified features for an incoming message.

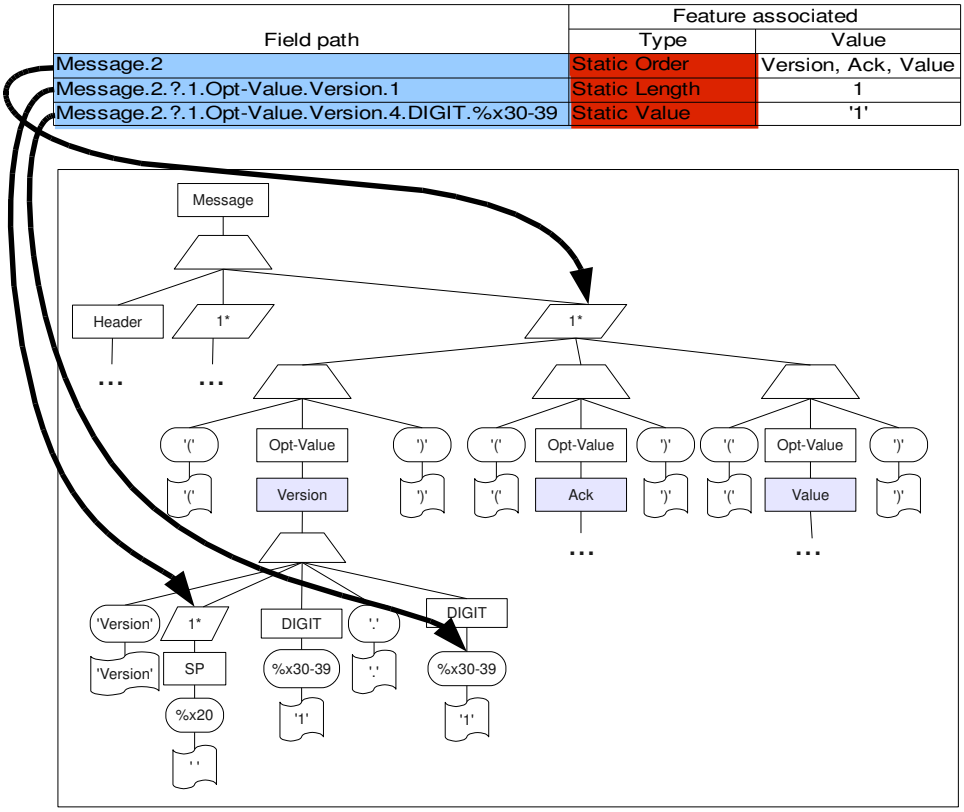


Figure 6.7: Features identification

Once a set of distinctive features is obtained, some well known classification techniques can be leveraged to implement a classifier. We have leveraged the classification technique in a k-neighbor approach where the device with bigger score is considered as the matching device.

6.4 Optimization Issues

This section presents two rules of thumbs for optimizing the training and as consequence, the performance of the fingerprinting system. The first rule is described in section 6.4.1 which maximizes the classification of fields out of the training set of messages. The second rule described in section 6.4.2 modifies the parser to avoid possible problems with delimiters.

6.4.1 Classifying the Fields

Section 6.3.1 described how the system automatically discovers the classification of each field in the grammar. This process depends on the comparison of the messages from the same device, described in section 6.2. Whenever a repetition is found while comparing messages, a permutation between the children is searched. This permutation was based in the resemblance scores, which represents the best matching alternative for the children nodes in the message comparison. However, in this phase the interest is not to find the best match subtree but to classify as many fields as possible. To overcome this issue, in contrast to use the best permutation for comparison of children, all of the possible permutations are used, in this way the scope for classifying items radically increases.

6.4.2 Optimizing the Parser

The reader may have noted that in the Toy grammar illustrated in figure 6.2 (a) the presented delimiters were "(" and ")" (parenthesis). This has been chosen for the sake of simplicity and was used to avoid exposing a leak of signature information during the explanation of the fingerprinting. Assume, the new rule "Message" to replace the rule defined is:

```
Message= Header 1*SP Opt-Value *(';' Opt-Value)
```

In fact, it does not represent an important change in how the messages are written but it does represent an important fact in the behavior of our fingerprinting approach, even so, only the delimiter was changed. We can observe that the first "Opt-Value" in the message is not reduced into the repetition. As a consequence, if the order of *Opt-Value* items in the message is changed and this involves the first item, the fingerprinting system won't be able to detect it as it does not belong to the repetition.

To solve such problems, when the parser is built, a search for delimiters is done simultaneously. This search consists in finding all the repetition productions in the grammar in which their first (or last) items represent some type of delimiters. A delimiter is found in the beginning (ending) of a repetition if the right-most (left-most) sibling of the repetition is the same production than the one that immediately follows the delimiter. Using the above "Message" rule, the *';'* is a delimiter because "Opt-Value" is a sibling and the right-most child of the repetition. For each delimiter found, the rule is modified in order to fulfill the leaking problem by joining the item inside the repetition as a choice between them. This new rule accepts a bigger language but will still be able to match the original message causing no problem to the fingerprinting algorithms. Taking as an example the Message rule, the modification will be as follow:

```
Message= Header 1*SP *(';' / Opt-Value)
```

6.5 Conclusion

In this chapter we described a novel approach for generating fingerprinting systems based on the structural analysis of protocol messages. Our solution automates the generation by using both formal grammars and collected traffic traces. It detects important and relevant complex tree like structures and leverages them for building fingerprints. It is different than the approach of H. Yan [145] since we do not require manual observation of the messages in order to extract the signatures and our process classifies message per message independently, using a passive approach. The applicability of our solution lies in the field of intrusion detection and security assessment, where precise device/service/stack identification are essential. Our work is relevant to the tasks of identifying the precise vendor/device that has generated a captured trace. We do not address the reverse engineering of unknown protocols, but consider that we know the underlying protocol. The current approach does not cope with cryptographically protected traffic. A straightforward extension for this purpose is to assume that access to the original traffic is possible. Our contribution consists in a novel solution to automatically discover significant differences in the structure of the syntax messages. As a result it allows us to classify message per message independently of the previous ones.

Chapter 7

Experimental Results

We have implemented the Fingerprinting Framework approach in Python. A scannerless GLR parser (Generalized Left-to-right Rightmost derivation parser) has been used (Dparser[42]) in order to solve ambiguities in the definition of the grammar.

We have instantiated the fingerprinting approach on the SIP [120] protocol. The SIP messages are sent in clear text (ASCII) and their structure is inspired from HTTP. Several primitives - REGISTER, INVITE, CANCEL, BYE and OPTIONS - allow to perform session management for a voice/multimedia call. Some additional extensions do also exist -INFO, NOTIFY, REFER, PRACK- allowing to perform presence management, customization, vendor extensions etc.

We have captured 21981 SIP messages from a real network, summarized in table 7.1. These message belong to 26 different implementations, considering also that some of them are different versions of the same implementation. Indeed, the collected traces belong to different networks with different configurations from each of the devices containing a wide diversity of messages.

It was sufficient to train the system with only 15% of the 21981 messages. The training set of traces for each device is proportional to the number of messages generated by that device, however they were randomly sampled. Finally, the system was tested to classify all the messages from the set.

The training phases has been easily parallelized. We did deploy it using 150 computers from the Grid5000 platform[43]. The computers were Xeon-Woodcrest, dual-core 64 bits with 2GB RAM. Table 7.2 shows the efficiency of each phase of the process.

271 features were discovered among all the different types of messages. It is worth noting that the features are not linked to the type of message but to the fields of them. For instance, even if a specific type of message was never seen before, the system will be able to classify it based on the common structures shared between the messages. Thus, the features represent items order, different lengths and values of fields where non protocol knowledge except its syntax grammar had been used. Between two different devices the distance of different features ranges between 35 to 198 features, where most of the lower values correspond to different versions of the same device.

The classification process usually recognized between 10 and 58 features in one message and it takes in average 0.06 seconds to classify a single message. We consider that this number can be improved if a more powerful language (like C) is used instead of Python.

To validate our approach we used the trained system to classify all the collected traces. We keep the following constrains to evaluate our classifier:

[42] `http://dparser.sourceforge.net/` last checked on December 2008

[43] Experiments presented in this manuscript were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see `https://www.grid5000.fr`)

Device	Software/Firmware version
Asterisk	v1.4.4
Cisco CallManager	v5.1.1
Cisco 7940/7960	vP0S3-08-7-00
	vP0S3-08-8-00
Grandstream Budge Tone-200	v1.1.1.14
Linksys SPA941	v5.1.5
Thomson ST2030	v1.52.1
Thomson ST2020	v2.0.4.22
SJPhone	v1.60.289
	v1.60.320
	v1.65
Twinkle	v0.8.1
	v0.9
Snom	v5.3
Kapanga	v0.98
X-Lite	v3.0
Kphone	v4.2
3CX	v1.0
Express Talk	v2.02
Linphone	v1.5.0
Ekiga	v2.0.3

Table 7.1: Tested equipment

Type of Action	Average time per comparison	Number or comparison computed	Total computed time(1)
Comparison for Phase 1	722 milisec	571.234	1 hour
Comparison for Phase 2	673 milisec	8.175.419	10 hours

(1) Experiments were carried out using the Grid'5000 experimental testbed

Table 7.2: Efficiency results obtained with the system

True Positives: Messages correctly classified (associated to their source) corresponding to devices analyzed during the training.

True Negatives: Messages classified as unknown corresponding to devices which were not used during the training.

False Positives: Messages classified to one of the known devices but were not generated from such a device.

False Negative: Messages classified as unknown by the system from devices which were used during the training.

Table 7.3 summarizes a weighted average obtained from the sensitivity, specificity and accuracy, computed individually from each device (whether our classifier identified it well or not). The sensitivity represents the proportion of correctly classified corresponding device over all the messages classified as such device.

$$Sensitivity = (true\ positives)/(true\ positives + false\ negatives)$$

Classification	True Positive 21422	False Positive 32
	False Negative 490	True Negative N.A.
Accuracy 0.998	**Sensitivity** 0.976	**Specificity** 0.999

Table 7.3: Accuracy results obtained with the system

The specificity represents the proportion of messages correctly classified as not being the specific device over all the messages classified as not being such device.

$$Specificity = (true\ negatives)/(true\ negatives + false\ positives)$$

The accuracy represents the proportion of correct classifications.

$$Accuracy = \left(\frac{true\ positives + true\ negatives}{true\ positives + false\ positives + true\ negatives + false\ negatives}\right)$$

In table 7.3, we can observe that the results are very encouraging since the specificity is high as well as the accuracy, however some observations made by deep analysis of the misclassified messages are described below.

Figure 7.1 illustrates the messages which were misclassified. From what it can observed, 559 messages - just 2.5% of the total set of messages - were not successfully identified. The causes of misclassification are:

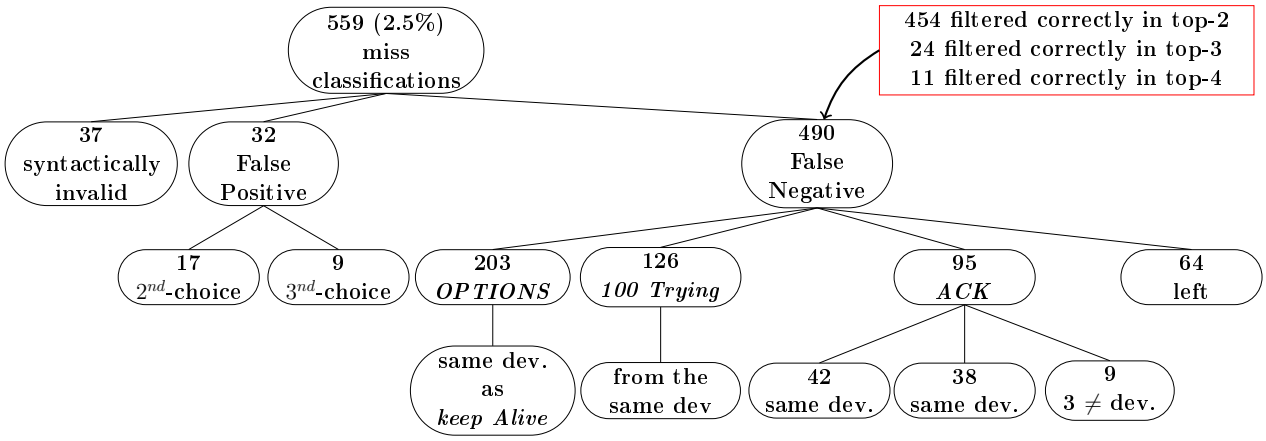


Figure 7.1: Miss-classification analysis

Syntactically Invalid: there are minor implementation issues which do not allow to parse any message non-compliant with the underlying grammar (e.g. bugs, relax checking).

False Positives: just a few of the messages were associated to a wrong device. These where messages usually forwarded by a proxy, thus they contained both signatures (the source entity and the ones from the proxy).

False Negatives: this vast majority represents those messages for which the system did not match enough signatures to make a successful classification. However, it is worth noting that the system was able to filter the possible sources to 2 devices (in 454 messages), 3

```
OPTIONS sip:192.168.1.4:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.101;rport;branch=z9hG4bKc0a801650000000b4550c64f0
Content-Length: 0
Call-ID: F28A8FE4-1FF9-4937-AF8D-81B29FD607FE@192.168.1.101
CSeq: 20 OPTIONS
From: <sip:0231555777@192.168.1.4>;tag=1286870423922
Max-Forwards: 70
To: <sip:192.168.1.4:5060>
```

Figure 7.2: OPTIONS message as *keep alive*

devices (in 24 messages) and 4 devices (in 11 messages) where the appropriate one was in the respective set. Also, from the set of False Negatives we can identify 4 groups:

OPTIONS: these 203 messages were generated by only one implementation and its purpose was to keep the firewall port open (as SIP was transported by UDP). Figure 7.2 shows the message in which there is almost no sign of possible signatures.

100 Trying: these 126 messages were generated by only one implementation as well. The message purpose is to temporally acknowledgement a request while it is being processed. For the case of this specific device, there was almost no content in the messages depending on specific configurations.

ACK: 95 messages of this type were miss-classified, most of them belonging to only 2 devices.

Others: 64 messages of diverse types for which no explanation was found were misclassified..

Finally, we created a set of messages which have been manually modified. These modifications include changing the User-Agent, Server-Agent and references to device name. As a result, deleting a few such fields did not influence the decision of the system, neither did it the alteration of the content of the banner implementation (e.g. modifying the User Agent field). However, as more modifications were done, less precise the system became and more misclassification were done. Also, we conducted a successful live demonstration at IPTComm 2008[44] where the public was able to modify messages in order to test the robustness of the system.

7.1 Training Scalability

A remaining open question was "How many messages we need to get a sufficient training?". This depends on how representative the messages used in the training are. Therefore we conducted experiments in which we wanted to observe how the classifier did improve as more data was used for the training. Therefore we took a subset of traces, i.e. 2091 messages belonging to 6 devices and we computed several trainings. The system has been trained 15 times using 15% of the messages, 5 times for each successive 10% and 1 time using the 100% of the messages.

Table 7.4 shows the collected results for the different trained systems.

Figure 7.3 illustrates those results in a ROC graph (or sensitivity vs (1 - specificity) plot). The red dotted line represents a random guess classifier. The closer the dots are to the Top-Left corner, the more accurate the classifier is. We can observe that all the different instances of the training fit in the Top-Left square which in fact give us a good feeling for the approach.

[44] IPTComm '08: Principles, Systems and Applications of IP Telecommunications, Heidelberg, Germany

Messages	Features	FP	FN	Accuracy
15%	125 ~ 189	3 ~ 23	35 ~ 231	0.997 ~ 0.979
20 ~ 40%	393 ~ 333	6 ~ 24	18 ~ 44	0.998 ~ 0.995
50 ~ 90%	345 ~ 389	1 ~ 2	20 ~ 19	0.998
100%	151	1	20	0.998

Table 7.4: Training details

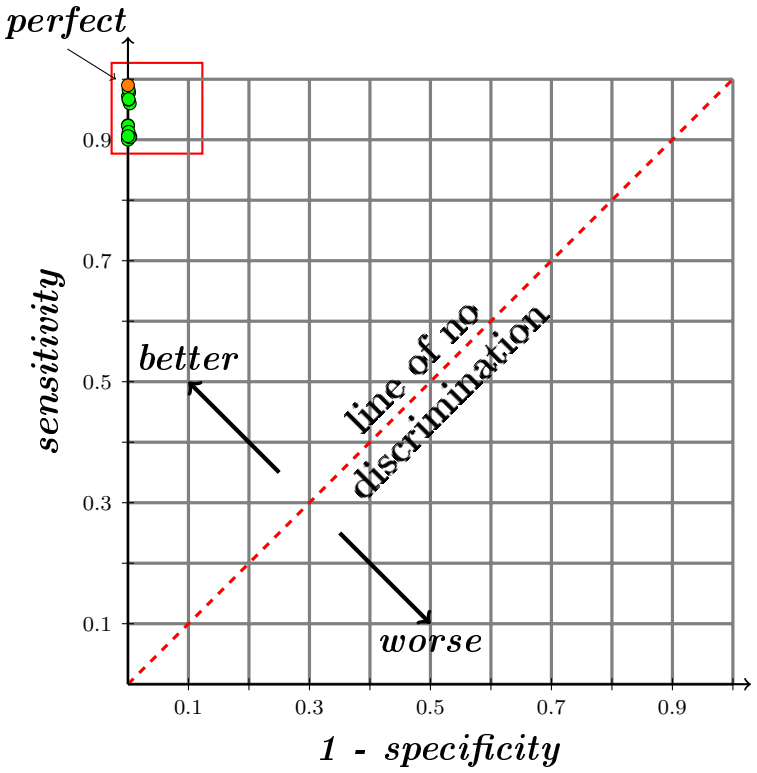


Figure 7.3: Training scalability

Figure 7.4 shows a closer look of the Top-Left square from the previous graph. The orange dot represents the system trained with the 100% of the traces and it is the optimal performance/accuracy that we can get with our approach. While the green dots represent each instance of the training done with only 10% of the traces randomly chosen. It can be observed that all the results are accurate but they proximity to the optimal training depends on how representative the set of chosen traces is.

7.2 Conclusion

In this chapter we described and evaluated the experimental results obtained by the implementation of our passive fingerprinting approach. The obtained results are very encouraging. We did not compared our approach with any other existing approaches for several reasons. First our approach is passive, comparing it with an active fingerprinting does not make sense since the objectives are different. We are working towards an approach capable of monitors a VoIP network without disrupting or invading the network with packages. Second, we did not consider

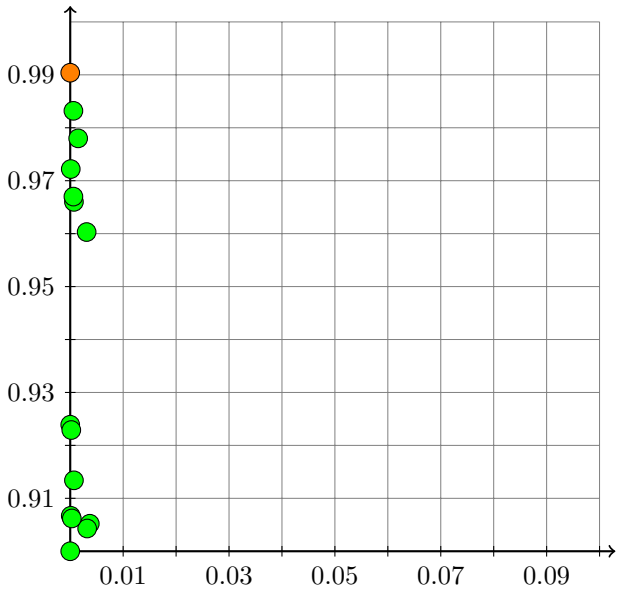


Figure 7.4: Zoomed training scalability

using passive approaches which classify the messages based on their banner (e.g. User Agent, Server) since in certain collected traces we expressly configure the devices to hide their banners. This simple fact will be sufficient to obtain awful results from such naive fingerprinting systems. Third, the only SIP passive fingerprinting system to our knowledge is [145], however its features are extracted via a manual observation, which make it not scalable (currently 13 devices). Indeed, the two strongest features found by their analysis relates to the order in which the Methods Allowed and Headers were presented. However, in our training we found that the order of Headers is not a strict parameter since several devices often switch their positions. In terms of the Methods Allowed, our system was equally able to identify it as feature given our approach a more robust set of features.

The system has demonstrated to be really accurate, however we consider that its performance can be highly improved if the approach is implemented in a different language, like C.

Part IV

Vulnerability Testing

Chapter 8

Fuzzing

Fuzzing is an important topic in the context of security assessment, software testing and black box testing approaches. The major idea behind fuzzing is that input data will be tampered with random payload and will be injected in order to test the data validation and processing of a target application. Several ways exist to generate the injected data. The data can be generated from scratch, by mutating existing valid data item or obtained by merging existing data. These possibilities do not represent the challenge itself, because random functions can always generate such data. The main challenge however resides in how to create input data that can reveal software errors and/or noncompliance to standards.

From the point of view of a tested device, two main logical structures can be considered (as illustrated in figure 8.1):

1. The unit which parses and translates a message to a structured format
2. the unit that will process that structured data and define the behavior to be executed

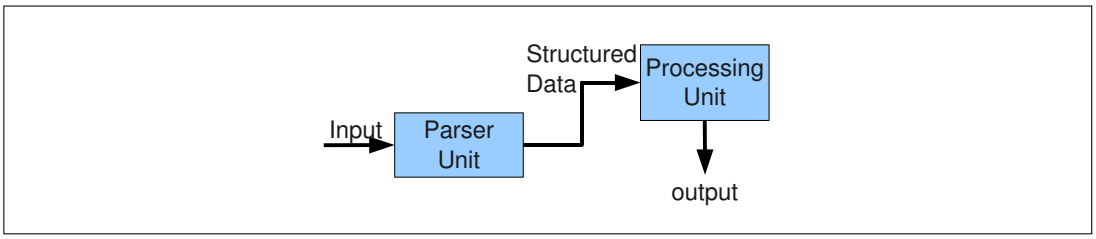


Figure 8.1: Device logical structures

Assuming the generic structures of Figure 8.1 a crafted message can be classified based on the effect on the target:

1. Messages that are not syntactically compliant and are therefore rejected by the parser in the target entity.
2. Messages that are not syntactically compliant, where the parser does not detect such irregularities, but however the processing unit rejects them.
3. Messages syntactically not compliant, where neither the parser nor the processing unit are able detect the irregularities.
4. Syntactically compliant messages containing semantic irregularities which are rejected by the parser.

5. Syntactically compliant messages containing semantic irregularities that are rejected by the processing unit.

The first and fifth types are directly considered as garbage because they do not have any effect on the tested entity. The second type instead, allows the message to be processed but its failure is detected in an upper layer. The message itself did not provoke a a fault in the target device but reveals a potential security hole. New messages may dig further to find more serious problems in the corresponding unit. In case of the third type, two consequences might arise: either a vulnerability is found or the information crafted in the message is not of concern for the entity. The latter may be the case of a proxy, which usually ignores the fields that are of no interest to speed up the process. The fourth type is associated with messages that may restrict the interoperability with other devices.

Our first objective was to define a flexible technique capable to generate messages of any of these types. We wanted to do more than current fuzzers do, which in most cases are restricted to simple text based substitutions of large data chunks and/or injected format string attacks required to test for common buffer and format string vulnerabilities. This work derived in the following publications [10, 14].

Some of the existing fuzzers use simplistic operational models, while others provide a rather complex interface, requiring major work to adapt them for additional tests. This last issue was one driving force in our work. We decided to research how complex fuzzers can be build on top of a small set of evolving and adaptive key building blocks.

Our aim was to provide a self-learning fuzzer that can evolve and use structural domain specific knowledge. Evolution is a key design feature required to built smart protocol fuzzers, while a domain specific knowledge is a starting point for obtaining better results.

The second challenge that we addressed was how to evaluate the effectiveness of generated fuzzed input. For this issue, some ideas can be found in the research papers on fuzzers and software testing [25, 90, 100, 124, 135]. The major issue is how to automatically detect that a fuzzed message was successful. If a device crashes, then probably checking its status before the reboot, might detect the crash. Checking the on-line status for embedded devices is not sufficient. There is a need for more complex approach where the target equipment is check at all time even while the test is been executed.

Our final interest was set by the idea to be able to fuzz at a protocol behavior level rather than only syntactically.

8.1 Fuzzing Framework

The test validation process described in this chapter is illustrated in Figure 8.2. It is composed of two autonomous components, the Syntax Fuzzer and the State Protocol Fuzzer, which jointly provide a stateful data validation entity. A test is generated by a scenario, where the scenario is a set of directives which represents a high level goal. Note that one scenario can generate several different tests, where each test is an instance of the directives used. Scenarios are based on both protocol specific domain knowledge and random data injection. As well, based on their functionality, scenarios can be divided in two: syntax and protocol scenarios. For instance, a protocol scenario tests SIP verbs (*INVITE, REGISTER*, etc. transactions) while a syntax scenario is useful for testing SIP fields (*From, To, Content-Length*, etc.).

The tests may be similar to the normal behavior or can flood the device with malicious input data. Such malicious data can be syntactically non compliant (with respect to the protocol data units), or contain semantic and content wide attack payload (buffer overflows, integer overflows, formatted strings, or heap overflows).

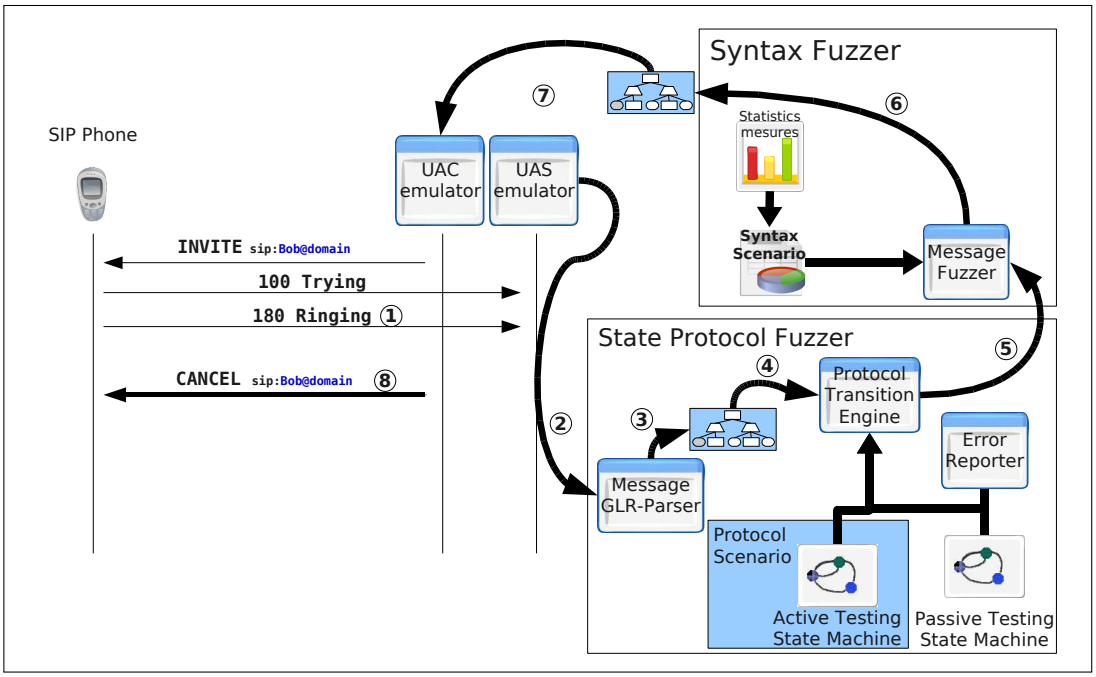


Figure 8.2: Fuzzing framework

The Syntax Fuzzer takes a syntax scenario and the provided ABNF syntax grammar to generate new and crafted messages. The syntax scenario drives the generation of the rules in the syntax grammar and may also depend on the protocol scenario in order to generate the final message (appropriated or not) to be sent to the target entity.

The State Protocol Fuzzer component does passive and active testing. Therefore, two state machines are required: 1) one specifying the SIP state machine and 2) one specifying the testing state machine.

The first state machine is used for the passive testing and controls if there is any abnormal behavior coming from the target entity during the execution of the tests. This state machine can be manually specified or directly inhered from SIP traces of the target entity; thus its behavior is induced after normal circumstances.

The second state machine is used for the active testing and it is the one that drives the behavior of the test. This state machine is defined by the user and it can evolve according to events triggered in the tests.

Figure 8.2 also shows the functional framework of the approach, where in the example our framework initiates a session by sending an INVITE. Our User Agent Server (UAS) processes the message and informs the Protocol Transition Engine. The latter checks with the protocol scenario being used and induces the incoming or outgoing message that should follow. If it is required by the protocol scenario to generate a new message, then it will be constructed by the Syntax Fuzzer following specific parameters given by the protocol scenario. This message is constructed by the Syntax Fuzzer according to the defined rules in the syntax scenario. Note that if the protocol scenario decides that another message should be received in order to proceed, the State Protocol Fuzzer will remain idle.

The traditional approach in the fuzzing community is based on data input validation. This is done by generating crafted messages and observing the resulting behavior in the target entity. Generally, the resulting behavior is observed in terms of "aliveness factors" (i.e. state of the device: crashed or functional). With the help of the Error Reporter component we can extend the analysis by observing incorrect transitions over the states and observe responses which are

not syntactically compliant.

8.1.1 Stateful Fuzzing Evaluation

Three techniques to evaluate the effectiveness of crafted messages in a target entity have been investigated in our work, each one of them can be considered as different protocol scenarios in which they interact with each other over one test:

Learning Test : the normal behavior of the target entity should be learned for testing its aliveness in case it crashes during the tests. This alive behavior may be obtained by sending an *OPTIONS* message and observing its replies, if any. Some entities may not support or be configured to ignore such messages. For these cases another sequence of messages may be sent as *REGISTER* or *INVITE and CANCEL* to allow to learn the normal behavior. This sequence is sent several times before starting the testing in order to ensure that the entity replies always in the same manner. It is important to note that such messages are not crafted because their only purpose is to evaluate the aliveness and correct functionality of the target entity.

Active Test : the testing of the target functionality consists in a defined state machine with sequences of messages - see the Testing State Machine in figure 8.2. This state machine represents the scenario describing how the evaluator should react to specific events. It may describe the behavior after unexpected messages, timeouts or normal events. In the case where some transitions are not defined in the protocol scenario state machine, the passive testing state machine can take control of the test generation in order to properly finish the transaction.

Alive Test : finally, when the test is finished, it is also necessary to check if the device is alive as well as if it is behaving in a usual manner. Note that a test may finish by timeout which does not really mean that the device crashed, but that the crafted message was too incorrect to be replied to. For this case, every time a active test is launched, the alive test may try to detect that the target entity is either alive or that it is still coherent with its initial behavior observed by the learning test. Once this step is made, errors are either reported or it continues with a new active test.

8.1.2 Reporting Events

Events are reported in one of the following cases:

- If a message generated by the target entity is not compliant with the protocol syntax, i.e. the information was not well interpreted or was not considered at all, as it is the cases of some proxies. This is considered a good starting point to dig for vulnerabilities.
- If a message generated by the target entity generates a not recognized transition in the Passive Testing state machine, i.e. the last or previous crafted messages drive the target entity to a state where the protocol specification is violated.
- If the target entity generates a message which is not contemplated by the active state machine. This can be the case where the test is willing to avoid certain behaviour, for instance bypassing the authentication challenge.
- Finally, when the aliveness tests are not responding as they should, either because no answer at all is obtained from the target entity or if a different one with respect to the already learned one is got.

8.2 Syntax Fuzzing

Different approaches propose interesting techniques for fuzze the syntax of the message [17, 38, 51, 68, 83]. However, they fail in the sense that messages need to be created from a seed or, if not, most fields of the messages are either static, semi-random or do not consider the protocol syntax as a point of reference. In our approach we use a context-free grammar to generate the messages but it is complemented with a parser for the same grammar. Therefore, the parser can extract information from previous messages and thus guide the fuzzer to generate the message fields based on measured statistics. The latter will give the fuzzer a wider set of seed for each specific field, where the quantity of seed will directly depend on how representative were the traces and not just on some specific values given by the fuzzer designer.

Our syntax fuzzer approach supports different procedures in order to build a message: 1) create the messages from scratch, 2) mutating some seed messages (or fields) or 3) merging fields of different messages in order to build a new one. Even though, mutation is considered less effective than generation [105], it is worth noting that all trials have been performed in naive mutation fuzzers. This fuzzers where mostly random fuzzers and did almost have no knowledge about the protocol been fuzzed. Thus, their efficiency were reduce to simple bit mutations. Therefore, we expect better results from a fuzzer which can combine the same techniques for generating messages from scratch or by mutation where it benefits from a solid protocol background knowledge.

Fuzzers are often classified based on multiple criteria: their speed to generate messages, their capability to discover vulnerabilities or by the number of substitutions supported. We consider in this chapter a more formal approach where a Fuzzer Expression Grammar is defined in order to describe the coverage of randomness in the generated message. We consider that this approach is capable of describe all possible mutations of the fields involved in the message syntax. This definition is closely related to the Parsing Expression Grammars [63], which formalizes the parsing grammar concepts.

Another important fact of fuzzers is related to the inputs that have to be provided in order to launch the test. Such inputs will define the generality, the specificity and the overall behavior. A certain type of grammar is required as well as knowledge about the syntax of the messages and the possible variable fields that may be changed by the fuzzer. Most of the time they require a lot of information and cover only a small scenario of generated data. It is also hard to know if the compliance with the protocol is kept or not. Very often, the rules to randomize such fuzziness may be either too simplistic and limited or too complex to be used in the creation of new tests.

Our syntax fuzzing approach takes two inputs. The first is a ABNF (Augmented Backus-Naur Form) grammar [41], which is the standard syntax definition of a protocol specification. Thus, the fuzzer provides the flexibility to be adapted to different protocols. It is capable to generate messages compliant or not with the underlying grammar based on the second input: the Syntax Scenario..

8.2.1 Fuzzer Expression Grammars

A Fuzzer Expression Grammar inherits from an ABNF grammar, being inherently linked to the underlying grammar. An additional syntax evaluator exists in a Fuzzer Expression Grammar. This syntax evaluator is the entity guiding the reduction of the rules in order to create a new message. As it will be explained later, this process may decide whether to be compliant or not with the syntax of the grammar.

A Fuzzer Expression Grammar consists of a 5-tuple $G = (\Sigma, N, P, E, n_0)$ where the components Σ, N and P represent the set of Terminals, Non Terminals and Production rules (respectively), the n_0 represents the non Terminal rule to be reduced and the syntax evaluator E

is:

E = Syntax fuzzer evaluator of the form $E : e \times \theta \rightarrow \Sigma^*$ where e is the *grammar expression* (as defined in section 6.1.1) to be fuzzed and θ is the environment state. Thus, the syntax evaluator concludes the evaluation when a sequence of Terminals have been obtained.

A message m generated by this fuzzing grammar is

$$m = E(n_0)$$

thus, the typical objective of such a message is to represent a data input validation test for a protocol implementation instance.

8.2.2 Expressive Power

In order to formalize the expressiveness of the approach, an evaluation interface is defined in six main functions:

- $\mathcal{T} : \Sigma \times \theta \rightarrow e \times \theta$, which may replace a Terminal by another item.
- $\mathcal{N} : N \times \theta \rightarrow e \times \theta$, which may replace a Non-Terminal by another item.
- $\mathcal{C} : e_1 \ / \ .. \ / \ e_n \times \theta \rightarrow \mathbb{N}_{\{1,n\}}$, which decides which item index should be chosen.
- $\mathcal{R} : e^{(i,j)} \times \theta \rightarrow \mathbb{N}_{\{i,j\}}$, which decides how many repetitions should be reduced.
- $\mathcal{S} : e \times \mathbb{N} \times \theta \rightarrow e \times \theta$, which may replace the i-item of the Sequence by another item.
- $\mathcal{I} : e \times \mathbb{N} \times \theta \rightarrow e \times \theta$, which may replace the i-repetition of the Repetition by another item.

All these components interact with the syntax evaluator in order to generate a new fuzzed message:

$$E, E_1 : e \times \theta \rightarrow \Sigma^*$$

$$E_1(e,\sigma) = \begin{cases} e & \text{if } e \in \Sigma^* \\ E(e,\sigma) & \text{otherwise} \end{cases}$$

$$\begin{array}{ll} E(e,\sigma) = E_1 \circ \mathcal{T}(e,\sigma) & \text{if } e \in \Sigma \cup \{\varepsilon\} \\ E(e,\sigma) = E_1 \circ \mathcal{N}(e,\sigma) & \text{if } e \in N \\ E(e_1 \ .. \ e_n,\sigma) = E_1 \circ \mathcal{S}(e_1,1,\sigma) & \\ \qquad\qquad .. \ E_1 \circ \mathcal{S}(e_n,n,\sigma) & \\ E(e_1/../e_n,\sigma) = E(e_i,\sigma) & \text{where } 1 \leq i \leq n \text{ and} \\ & \ i = \mathcal{C}(e_1 \ / \ .. \ / \ e_n,\sigma) \\ E(e^{(i,j)},\sigma) = E_1 \circ \mathcal{I}(e,1,\sigma) & \\ \qquad\qquad .. \ E_1 \circ \mathcal{I}(e,k,\sigma) & \text{where } k = \mathcal{R}(e^{(i,j)},\sigma) \end{array}$$

For the functions $\mathcal{T}, \mathcal{N}, \mathcal{S}, \mathcal{I}$ of the syntax evaluator, five operations were defined that can help to progressively construct the Σ^* based in the Fuzzer Evaluation itself. These operations are described below:

- Produce either a fixed string or a random one generated from a regular expression.

- Append *expression* productions generated by another syntax evaluator.
- Generate any rule defined by the grammar, which may be not the ones allowed in the current reduction.
- Generate a new rule defined on the fly, allowing the evolution of rules or addition of new ones.
- Generate a function rule. A function is a special case, because it escapes from the syntax concepts to define semantic actions. It is evaluated after the whole message has been reduced to Σ^* union other functions. Based on other items generated for the message, it generates the Σ^* appropriated for its current field. This function can be useful to add fields like checksum, content lengths, etc. However, infinite recursion has to be prevented.

The reduction of the expressions proceeds in a Depth First Search (DFS), where the generated message may be viewed as a tree (Figure 8.3), such that all the internal nodes are non-Terminal (e_5), Choices (e_2), Sequences (e_1 and e_4) or Repetitions (e_7) items and the leafs are Terminals (e_3, e_6 and e_8) or functions before being evaluated (e_9, where the functions set is denoted as F in the figure). In this way, each reduction branch can be uniquely identified in the tree by the path from the root to the current position. Definition 3 formalizes the *reduction path* concept.

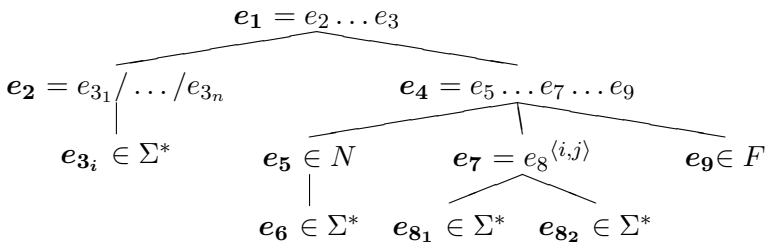


Figure 8.3: Tree reduction

Definition 3. *A reduction path, $x \triangleright xs$, will define the steps for which an expression reduces to another (i.e. from an expression e_i to arrive to the expression e_j). Each step is defined by the relation $\Rightarrow_F$ as:*

$$
\begin{array}{ll}
(e, x \triangleright xs) \Rightarrow_F (\mathcal{T}(e), xs) & \text{if } e \in \Sigma \cup \{\varepsilon\} \text{ and } \mathcal{T}(e) = x \\
(e, x \triangleright xs) \Rightarrow_F (\mathcal{N}(e), xs) & \text{if } e \in N \text{ and } \mathcal{N}(e) = x \\
(e_1 \;..\; e_n, x \triangleright xs) \Rightarrow_F (\mathcal{S}(e_x), xs) & \text{if } 1 \leq x \leq n \\
(e_1 / .. / e_n, x \triangleright xs) \Rightarrow_F (e_x, xs) & \text{if } 1 \leq x \leq n \\
(e^{(i,j)}, x \triangleright xs) \Rightarrow_F (\mathcal{I}(e, x), xs) & \text{if } i \leq x \leq j
\end{array}
$$

and it is said that the reduction path $x \triangleright xs$ success from e_i to e_j if the $\Rightarrow_F$ closure is equal to

$$(e_i, x \triangleright xs) \Rightarrow_F^* (e_j, [])$$

8.2.3 Example Evaluators

In this section, we will look at two example syntax evaluators for generating messages compliant to the underlying grammar, and to generate messages based on the merging of different other messages.

Compliant Grammar Evaluator

A definition of the inner functions of E, which randomly creates well formatted messages according to the specified grammar is illustrated below:

$$E : e \rightarrow \Sigma^*$$

$\mathcal{T}(e) = e$	
$\mathcal{N}(e) = P(e)$	
$\mathcal{C}(e_1/../e_n) = i$	where $1 \leq i \leq n$ chosen randomly
$\mathcal{R}(e^{(i,j)}) = k$	where $1 \leq i \leq k \leq j$ chosen randomly
$\mathcal{S}(e, i) = e$	
$\mathcal{I}(e, i) = e$	

Since the items contained in $\mathcal{T}$ create static strings, the ones from $\mathcal{N}$ invoke fixed rules, the ones from $\mathcal{S}$ represent an static order of items and $\mathcal{I}$ are specific items of the grammar, it is clear that they can not be used to modify the reduction of the message (if our intentions are to keep the generated message compliant with the grammar). Thus, the only possible randomness in a compliant syntax evaluator E are the functions $\mathcal{C}$ and $\mathcal{R}$.

Merging Messages Evaluator

A more complex syntax evaluator where a message is generated out of the composition of several messages (also maintaining its compliance with the underlying grammar) is showed here. We begin by defining the following function.

Definition 4. *Assuming that M represents the set of messages compliant with the grammar (e.g. those that may have been generated by the Fuzzer Evaluator), and P is the set of all possible reduction paths from all the expressions presented in such messages, the function ρ of the form*

$$\rho : M \times P \rightarrow e \cup \{\varnothing\}$$

obtains, if success, the corresponding expression for the reduction path, $xs \in P$, starting from the root expression of the message $m \in M$ that

$$(root(m), xs) \Rightarrow_F^n (e, [])$$

otherwise, if none expression exists, it returns $\varnothing$.

As a consequence, to allow the generation of messages out of the composition of others, the environment state of E is defined to be $\theta = M \times P \times P$. The variables $\tau, \delta \in P$ will represent the *reduction paths* from the initial and last triggered rule respectively. Assuming the variables ψ and ξ to be like:

$$\psi = \delta \quad \vee \quad \psi = \tau +\!\!+ \delta$$

$$\xi \in \{e \mid \exists\, m \in \omega : e = \rho(m, \psi) \wedge e \neq \varnothing\}$$

the definitions of the inner functions are detailed below.

$$
\begin{array}{ll}
\mathcal{T}(e,\omega,\delta,\tau) = (\xi,\omega,\delta \lhd \xi,\tau) & \\
\mathcal{N}(e,\omega,\delta,\tau) = (e,\omega,e,\tau \lhd \delta) & \\
\mathcal{C}(e_1/../e_n,\omega,\delta,\tau) = i & \text{where } 1 \leq i \leq n \\
 & \text{and } e_i = \xi \\
\mathcal{R}(e^{(i,j)},\omega,\delta,\tau) = k & \text{where } 0 \leq i \leq k \leq j \\
 & \text{and } k = length(\xi) \\
\mathcal{S}(e,i,\omega,\delta,\tau) = (\xi,\omega,\delta \lhd i,\tau) & \\
\mathcal{I}(e,i,\omega,\delta,\tau) = (\xi,\omega,\delta \lhd i,\tau) &
\end{array}
$$

Is it worth noting that when replacing the expression e by ξ, the reduction still being compliant with the underlying grammar, because ξ is reduced from the same rule under another context. However, the value of ψ chosen will define a degree of fuzziness in the resulting message due to the complete or relative path location of the expressions.

8.2.4 Learning Techniques

A feature not considered by syntax fuzzers is the fact of learning from observed messages and use this knowledge as a base of smart fuzzing.

To illustrate the importance of such technique, we consider the following toy grammar and use the random evaluator previously described in section 8.2.3 to generate fuzzed fields.

```
username = alphanum *(alphanum/"-"/"_"/"%"/"&")
alphanum =  ALPHA / DIGIT
ALPHA    =  %x41-5A / %x61-7A                    ; A-Z / a-z
DIGIT    =  %x30-39                              ; 0-9
```

It can be assumed that the priority by which items appear in a *username* consists in letters, number and then special symbols. However, a possible reduction of such grammar may look like:

$$username \rightarrow \text{d-\&\%3\&\%-\&q}$$

For this example, the syntax evaluator is up to generate a *username*, and when reaching the second item of the sequence, it has to decide among five choices, giving a low priority to numbers and letters. Thus, a message with high dimension and often invalid is more probable to be rejected by the target entity [136].

For this cause, the two interfaces below had been defined to provide some methods to generate "smart" syntax evaluators:

- Record Choice Indexes
- Record Repetition Lengths

Both interfaces receive as input the *sequence reductions* from the first rule, allowing to record statistics of repetition length and chosen items according to the function. Note that only these two methods are sufficient, because they are the only ones that can modify the evaluation flow of compliant messages.

8.3 Stateful Fuzzing

The State Protocol Fuzzer described in this section is targeted at the SIP protocol and it does passive and active testing. Domain specific knowledge is needed for this issue due to the fact that the fuzzer has to be able to distinguish between correct and incorrect behavior.

8.3.1 Passive Testing

Passive testing is performed in our framework based in monitoring all the traffic directed to the target entity and comparing with the known and allowed transitions in the underlying state machine. This underlying state machine is a specification of the states and transitions that represent a SIP implementation. This specification, beyond how it was conceived, will be used to analyse and determinate inconsistencies or abnormalities in the proper behavior of the target device.

The underlying protocol state machine on which the monitoring relies can be provided in two ways:

- a fully detailed state machine as specified by the standards
- a state machine induced from a sample of messages.

The latter approach was chosen in this research as an starting point since different implementations do not really work in the same manner. Subtle differences have been found in our experiments, where devices react differently to specific events. Even if a device does not behave as expected by the protocol specification, this does not mean that a vulnerability was found. Therefore, in order to evaluate the impact of the crafted input, the normal behavior of the target entity should be known a priori. Note that general behavior could also be used for the passive testing, but we risk of missing some device-specific anomalies.

SIP messages follow a hierarchy where Dialogs and Transactions are identified during a session. A dialog is uniquely identified by the *Call-ID* and a local and remote tag; such tags are presented in the *From* and *To* headers. Meanwhile, a transaction is identified by the *CSeq* header and the *Via Branches* of the top most *Via* header located in the message. Thus, a transaction belongs to only one dialog, but the latest may have many transactions. Also, a dialog is kept between two entities, even in the case where more entities are involved in the session.

To assume a simple model, the state machine is conceived just for SIP transactions rather than dialogs.

To induce the state machine we use the sample messages to fill the SIP Information Model described in Chapter 5.2. Assuming we want to induce the state machine of an INVITE transaction, the first step is to query the information model for all the SIP transactions in which their first message is of type INVITE. From each different message following the INVITE in any of the selected transactions, we create a new transition in the state machine. It is important to note that the states do not store more information than just the outgoing transitions. Thus, after all the messages for each SIP transaction have been observed, we obtain as a result a tree representation of the INVITE transaction (which can be converted to a state machine by merging common states). We have studied several algorithms to optimize the inducing process [27,103,121], however as SIP transactions are quite simplistic, this naive approach is sufficient for our approach. One important case is when a new SIP transaction is triggered in middle of an ongoing dialog (e.g. the CANCEL transaction is good example since it takes place in the middle of an INVITE transaction). Therefore, nodes in the state machine are provided with a forking feature which will allow them to initiate intermediate transitions at specific states. Thus, leading to interleave the transaction and in this way conclude a full SIP Dialog. Figure 8.4 illustrates an hypothetical state machine obtained from samples of INVITE transactions. Each transition represent either an outgoing message (!) or an incoming message (?) plus the specific type of message (Response-Code or Request method).

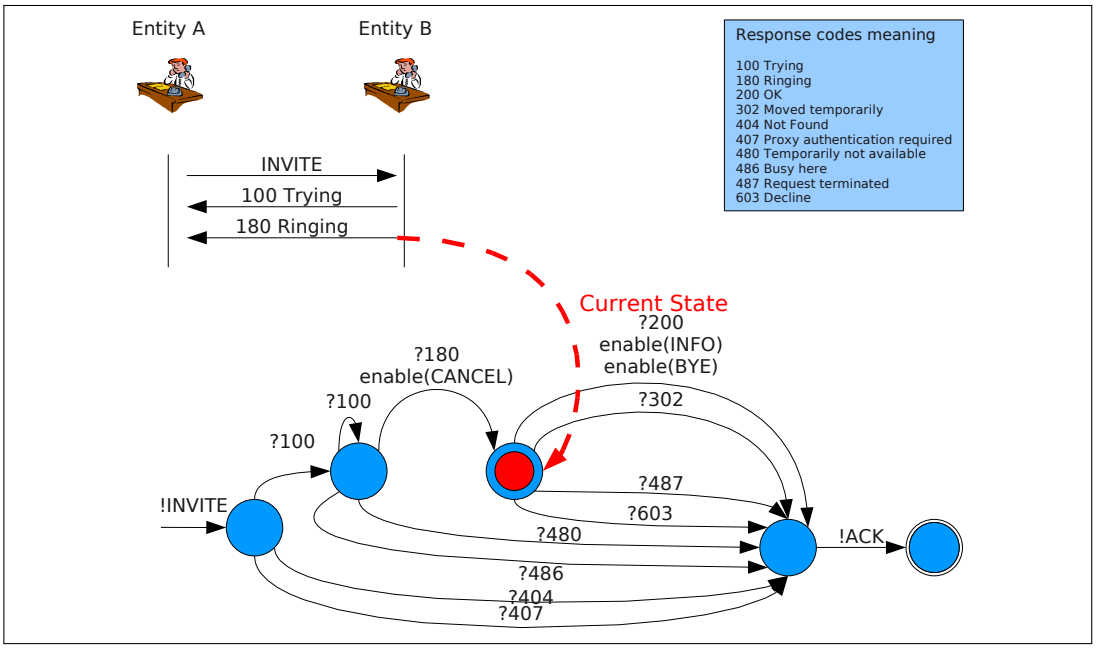


Figure 8.4: Learned state machine from an INVITE message
The transition with quotation marks (?) means that these messages correspond to the responding entity. The exclamation mark (!) means that the messages belong to the requesting entity.

8.3.2 Active Testing

The Testing State Machine of the protocol fuzzer is used to guide the testing by emulating malicious or normal behaviors. Such a state machine follows the principles of the Event-driven Extended Finite State Machine (EEFSM) described by David Lee et al. in [90]. However, in our approach, the above algorithm may not only be used to follow the system state but also to simulate and force one entity to perform different assessment operations.

An active test is composed of an unique SIP Dialog but with several SIP Transactions. Therefore, the test may have several state machines (one for each transaction). Thus, the test is associated to the current SIP dialog identifiers (i.e. the same Call-ID as the first message captured or generated which initialized the state machine) and then each outgoing/incoming message has to match such identifiers in order to be used in the test (unless explicitly said). The same applies to each state machine which in fact represents a SIP transaction and therefore they are tight to their identifiers (CSeq and Via Branch).

Then, if an arriving message matches the dialogs identifiers and the transaction identifiers of one of the state machines, the message is being processed by the state machine at its current state. Otherwise, if an outgoing message is expected to be generated, then only one of the outgoing transitions (denoted with !) matching the current state of the state machine will be considered as the source to generate the next message.

In the evaluation process the state machine should decide which will be the new outgoing message. This decision is computed from the testing state machine where it searches all the possible events that match the current state. Each state machine node has the following conditions and properties for its transitions:

- The type of the expected message (i.e. the message type may be a request method or reply code as *INVITE, CANCEL, 180 RINGING, 200 OK, etc.*),

- The direction of the message, either incoming or outgoing (Note that dialogs are between two entities, therefore the state machine keep that property),
- Pre condition to satisfy based in the environment of the state machine,
- Function to apply when the transition is selected,
- Enabled time of the transition. Either a fixed starting and ending period or functions that validate this period,
- Weighted transitions, in case more than one transition is applicable at a time.

If a transition triggers the generation of a new message, the Syntax Fuzzer is informed in order to create such new requested message, otherwise new incoming messages are expected.

Based on this process, the Protocol State Machine may also report the existence of unexpected behaviors (e.g. unknown transitions). Assume that one test is trying to establish a call without using the correct credentials, if the fuzzer succeeds in initiating such call then a flaw was encountered or the credential has been guessed. In either case, the protocol behavior will look normal but it will be worth to inspect what has been going on during the test.

Note that the emulated behavior of the Protocol State Evaluator may change from one transaction to another. This is happening, because in a same Dialog different transactions may be initiated by any of the entities. The only thing left to the scenario is the randomness to initiate a new transaction as well as the randomness to select among the possible set of message types to be send.

Figure 8.5 illustrates a protocol scenarios which consists in a INVITE transaction. Several syntax scenarios are defined which correspond to different aggressively in the methodologies for creating the fuzzed message.

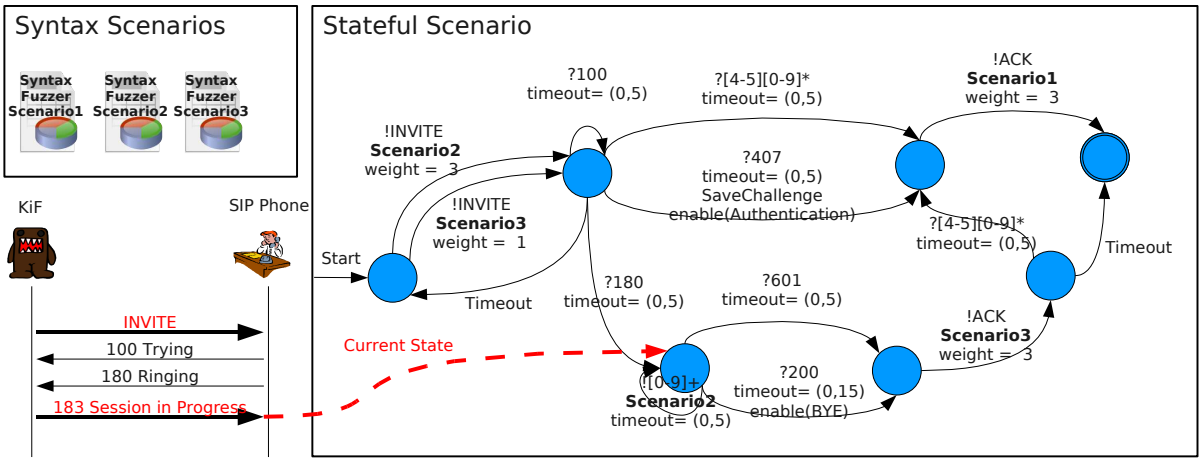


Figure 8.5: Stateful SIP scenario
The quotation (?) and exclamation marks (!) are as in Figure 8.4.
The *enable()* function creates a fork in the current state machine to enable the other state machine, meanwhile, information like Dialog ID and Transaction ID is transfer to keep the state awareness.

8.4 Fuzzer Effectiveness

The fuzzer may be configured to respond in different ways according to its defined fuzzer scenarios. Different types of behaviors have been proposed using the infrastructure defined in section

8.2 which showed the possible types of syntax scenarios behaviors.

The first and simplest, but however the less effective, is the random generation of the whole message. These messages are with certain probability compliant to the syntax provided by the protocol, but in fact, for the problem mentioned in section 8.2.4, they mostly are garbage that will simply be ignored by the target entity.

The second approach which results in more realistic messages is the mutation of existing messages. This mutation takes a set of fields in the original message that will be replaced by other values. Those latter values may be obtained by other rules, other choices or with random functions.

The third scenario consists in learning seeds from each field in the grammar from already existing messages. The seeds are stored and used following the directives of the syntax scenarios. Note that as the entity has now some knowledge about typical fields structure, the first technique will result in messages good enough to fool the target.

Finally, the last technique comes from the idea proposed by Lin Li et al. [93], where they described a self-assembly process for software components. A component is composed by a series of inputs and outputs, such that in order to assembly components their input/output have to be considered as if they were molecules. In fact, the assembly process is an analogy with real molecules, where they set the pressure, temperature and area to promote the assemblies. The idea of assembly fields of the message depending in their grammar inputs and output has been considered and tested during our experiments. The input of each field may be the global path from the root rule or the path from the current rule. While the output is all the items that may be reduced in the current field (i.e Terminals, Non-Terminals, Choice Index, number of Repetitions, etc). In this way, one field is attached to another generating a message that is a composition of many others. The function that defines which of the inputs may match with a given output is left to the interface defined by the fuzzer. In this way approaches for Genetic Algorithms (GA) may be defined.

Concerning the stateful testing, the protocol scenarios may evolve as well. Different environments may be defined based on the sequence of messages to test. Such sequences may be acquired from sample sessions or be manually constructed. In order to be able to reach to deeper states, it is also important to specify which messages are the ones that should be fuzzed, the time range for each type of response and the behavior after unknown events.

8.5 Conclusion

This chapter describes a stateful protocol fuzzer for SIP. The main contribution is a flexible, adaptive fuzzer capable to track the state of the targeted application and device. One of the components of our work is quite generic and reusable for any protocol for which an underlying grammar is known. The second one is dependent on the domain specifics (SIP). Our method is based on a learning algorithm where real network traces are used to learn and train an attack automaton. This automaton is evolving during the fuzzing process. We performed tests on VoIP phones and the results are promising (see chapter 9). We will continue our work by integrating other protocols, testing more devices (session border controllers, routers, media gateway controllers) and refining the learning/testing algorithms used in our framework.

Chapter 9

Experimental Results

We have built a tool implementing all the techniques described in chapter 8 (the tool is called **KiF** and it is available at `http://kif.gforge.inria.fr/`). The protocol syntax can be fuzzed very extensively, because the fuzzer can change every rule of the grammar. However this mechanism is only useful for fuzzing the syntax of a protocol. In order to fuzz state, the two state engines are used. One to represent the correct state, that an entity should be in and one to fuzz the testing entity with a wrong state. This mechanism however is completely independent from the functionality of the ABNF grammar. The passive state machine is induced from previous traces taken from the device while different active state machines were manually constructed having different goals in mind. Principally, the active state machines target different cases, some of them include:

- Normal flow of SIP Dialogs (INVITEs, REGISTERs, OPTIONs, SUBSCRIBESs, etc.) where the objective was to fuzze only specific messages in different state of the protocol. Such cases fuzze the message syntax diversity in two ways: 1) in random fields and 2) more smartly in chosen fields. The chosen fields depended in the goal of the test:
  - For testing the device's web server fields in the SIP message were fuzzed with typical XSS exploits and SQL injections (if the equipment had any SQL record support).
  - While testing the authentication mechanism, related fields to the challenge header where fuzzed. The fuzzed values were typical malicious input, lengthy values, non-existing/crafted options, invalid users (including SQL injection strings), etc.
- Unexpected/nonexistent messages at specif states where they include swapping the target role (i.e. changing the behaviour from UA client to server without consent). For such cases there is no defined behaviour in how the target must react, beside error responses. As error responses were not always the typical case, to continue a deeper testing in such invalid/unknown state, the active state machine was assessed by the passive state machine. The later report which were the typical behaviour after such messages assuming normal circumstances. This report was used to generate the next crafted message and continue the test until it reach a goal or receive an error message.

The most frequent vulnerability encountered during our testing [12, 13] is related to weak filtering of input data. This filtering does not properly deal with meta-characters, special characters, over lengthy input data and special formatting characters. Most of these vulnerabilities are due to buffer/heap overflows, or format string vulnerabilities. The most probable cause is that developers assumed a threat model in which VoIP signaling data would be generated only by legitimate SIP stacks. The real danger of this vulnerability comes from the fact that in most

cases, one or very few packets can completely take down a VoIP network. This is even more dangerous when realizing that in these cases the SIP traffic is carried over UDP, such that highly effective attacks can be performed stealthy via simple IP spoofing techniques.

Preventing these types of attacks at a network defense level is possible with deep packet inspection techniques and proper domain and application specific packet filtering devices.

Protocol tracking vulnerabilities go beyond simple input filtering of single messages. In this type of vulnerability several messages will lead a targeted device in an inconsistent state, albeit each message on its own does not violate the SIP RFC [120]. This vulnerability is caused by weak implementations of protocol state engines. Exploiting this vulnerability can be done in three main ways:

1. the device might receive inputs that are not expected in its current protocol state: for instance, when waiting for a ACK method, an 100 Trying is received,
2. the input might consist in simultaneous messages destined to different protocol states,
3. slight variations in SIP dialog/transaction tracking fields.

The discovery of such vulnerabilities is truly difficult. The fuzzing process should be able to identify whether a targeted device is not properly tracking the signaling messages and which fields can be fuzzed in order to detect it. The search space is in this case huge, being spread over many messages and numerous protocol fields, thus requiring smart driven fuzzing approaches [51,57,68,109].

The major danger with this type of attacks is that no application level firewall can completely track so many flows in real time and even in the case of known signatures, polymorphic versions of known attacks can be easily obtained and these will remain under the security radar. As of today, unfortunately no effective solution to prevent this type of attacks exists.

9.1 Security Advisories

This section describes some of the most remarkable vulnerabilities found during our testing. It is worth to know that the traffic generated by our tool discovered inconsistencies in the targets during the test, however a full analysis of each issue allowed us to identify the real vulnerability and the associated problem. For many of the performed tests we designed a fuzzing scheme. For instance, we wanted to conduct more complex scenarios like code injection. We wrote in the syntax scenarios malicious ABNF rules which can generate fields containing SQL and XSS injections. We taught the scenario possible fields where these rules may have some effect and we also let the scenario to choose other fields following some probabilities. The results of the test were encouraging and they are presented below, however external methodologies were used in order to determinate if the target was affected (e.g. browsing its web applications).

9.1.1 Weak Input Validation

The simplest vulnerability found was against a Thomsom ST 2030 VoIP phone (version v1.52.1) *CVE-2007-4753*. This attack consist in sending to the device a UDP message which does not contain any payload (i.e. an empty message containing zero bit in the data). This instance of an attack reflects the poor consideration taken for input generated maliciously.

9.1.2 Input Validation

Usually PBXs provide several VoIP services and gateways to the PTSN. Asterisk is one of them, an open source implementation widely used. However, in versions 1.4.1, 1.2.16 and older ones,

after receiving a crafted message the software crashes abruptly *CVE-2007-1561*. The message in this case is an anonymous INVITE where the SDP contains 2 connection headers. The first one must be valid, however the second must have an invalid IP address. Even more, the callee needs not to be a valid user. In cases where Asterisk is set to disallow anonymous call, a valid user and password should be used, and while responding the corresponding INVITE challenge the information should be crafted as above. This attack has been of a bigger magnitude due to the fact that a whole VoIP domain can be torn down by a unique message.

9.1.3 Attacks Against the Internal Network

Most VoIP devices have embedded web servers that are typically used to configure them, or to allow the user to see the missed calls, and all the call log history. The important issue is that, the user will check the missed calls and other device related information from her/his machine, which is usually on the internal network. If the presented information is not properly filtered, the user will expose her/his machine (located on the internal network) to malicious and highly effective malware. The following example discovered during driven fuzzing proves *CVE-2007-5411* affects the Linksys SPA-941 VoIP Phone (Version 5.1.8). This phone has an integrated web server which allows configuration and call history checking. A Cross Site Scripting vulnerability (XSS) [125] allows a malicious entity to perform XSS injection because the "FROM" field coming from the SIP message is not properly filtered. By sending a crafted SIP packet with the FROM field set for example to :

```
FROM: "<script x='" <sip:'src='http://attacker.com/beef/y.js'>@domain.org>;tag=1"
```

the browser is redirected to include a javascript file (y.js) from an external machine (attacker.com) as show in Figure 9.1. This external machine is under the control of an attacker and the injected javascript [125] allows to a remote attacker to use the victim's machine in order to scan the internal network, perform XSRF (Cross Site Request Forgery) attacks, as well as obtain highly sensitive information (call record history, configuration of the internal network), deactivate firewalls or even redirect the browser towards malware infested web pages (like for instance the underground tool MPACK[45]) to compromise the victim's machine. The major and structural vulnerability comes in this case, by the venture of two technologies (SIP and WEB) without addressing the security of the cross-technological information flow.

Figure 9.1: Linksys SPA-941 XSS attack

The impact of this vulnerability is very high : most firewalls/IPS will not protect the internal network against XSS attacks delivered over SIP. Additionally, users will connect to these

[45] `http://pandalabs.pandasecurity.com/blogs/images/PandaLabs/2007/05/11/MPack.pdf` last checked on December 2008

devices directly from the internal network and therefore the internal network can be compromised. Jeremiah Grossmann [125] showed how firewalls can be deactivated with XSS attacks and many other malicious usages do exist. Unfortunately, most VoIP devices have weak embedded WEB applications, such that other vulnerable systems exist and are probably exploited in the wild.

9.1.4 Stateful Crash

As it has been previously described, stateful fuzzing is more difficult to follow. Once a vulnerability has been encountered, it may be quite difficult to analyze the actual problem. One of such examples is the vulnerability found in the Nokia N95 cellphone which runs a VoIP client (RM-159 V 12.0.013) *CVE-2007-6371*. For this example, 2 dialogs are needed, the first one which set the device to an inconsistent state and the second dialogs that triggers the DoS in the whole cellphone.

Figure 9.2 shows the sequence of events which trigger this vulnerability. Basically, a CANCEL message arriving earlier than expected can turn the device into an inconsistent state which will end up in a Denial of Service attack.

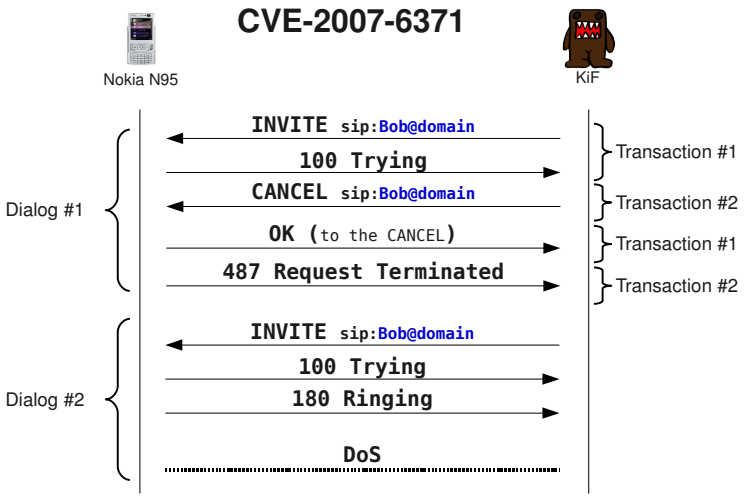


Figure 9.2: Nokia N95 DoS attack

9.1.5 Remote Eavesdropping Vulnerabilities

A rather unexpected vulnerability was discovered in the VoIP phone Grandstream GXV-3000 (v 1.0.1.7) *CVE-2007-4498*. Several SIP messages sent to the affected device put the phone off-hook without notifying the users. The attacker is thus capable to remotely eavesdrop all the conversations performed at the remote location. Figure 9.3 shows the messages exchanged by the attack. The impact if this vulnerability goes beyond the simple eavesdropping of VoIP calls, because an entire room/location can be remotely monitored by the loudspeaker integrated in the phone. This risk is major and should be considered when deploying any VoIP equipment. Although in the presented case, a software error was probably the cause, such backdoor left by a malicious entity/device manufacturer represent very serious and dangerous threats.

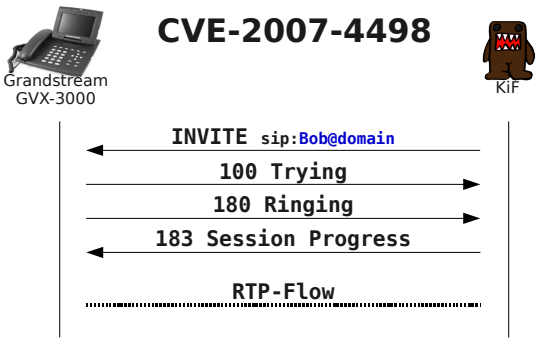


Figure 9.3: Grandstream GXV-3000 remote eavesdrop

9.1.6 Weak Cryptographic Implementations

The authentication mechanism in SIP is a standard shared secret and challenge-response based one (section 1.1.5). Nonces are generated by the server and submitted to an authenticating entity. The latter must use its shared key to compute a hash which is afterwards sent to the authenticator. This hash is computed on several values extracted from the authenticate message as well as the shared secret between the entities. The computed authentication is validated by the server and checked to authenticate a client. For efficiency reasons, very few server implementations track the life cycle of a valid token. In at least two Registrars/Proxies, Cisco CallManager and OpenSer (v5.1.1.3000-5 and v1.2.2 respectively) *CVE-2007-5468, CVE-2007-5469*, vulnerabilities where found where intercepted tokens could be replayed. These vulnerabilities are not simple man in the middle attacks, since intercepted tokens were reusable for long time periods. Even more authentication could be used for any other destination call due that the tested implementations do not allow to check if the provided URI in the Digest authentication header is the same as the REQUEST-URI of the message. However, from the SIP RFC [120]

> *"RFC 2617 [17] requires that a server check that the URI in the request line and the URI included in the Authorization header field point to the same resource. In a SIP context, these two URIs may refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the Request-URI in the Authorization header field value corresponds to a user for whom the server is willing to accept forwarded or direct requests, but it is not necessarily a failure if the two fields are not equivalent."*

it can be extracted that a server MAY check depending in the context of the call, which is not the case of any of the tested entities.

Figure 9.4 shows the flow of messages for such an attack. The impact of such a vulnerability is very high. Toll frauds and spoofing call identifiers are the straightforward consequences. The mitigation consists in trading off performance versus security and implementing efficient and secure cryptographic token management procedures.

9.1.7 Toll Fraud Vulnerabilities

Toll frauds occur when the true source of a call is not charged. This can happen by the usage of a compromised VoIP infrastructure or by manipulating the signaling traffic. It is rather amazing to see that although technology evolved, the basic conceptual trick of the 70's, where phreakers reproduced the 2600 Hz signal used by the carriers is still working. Thirty years after,

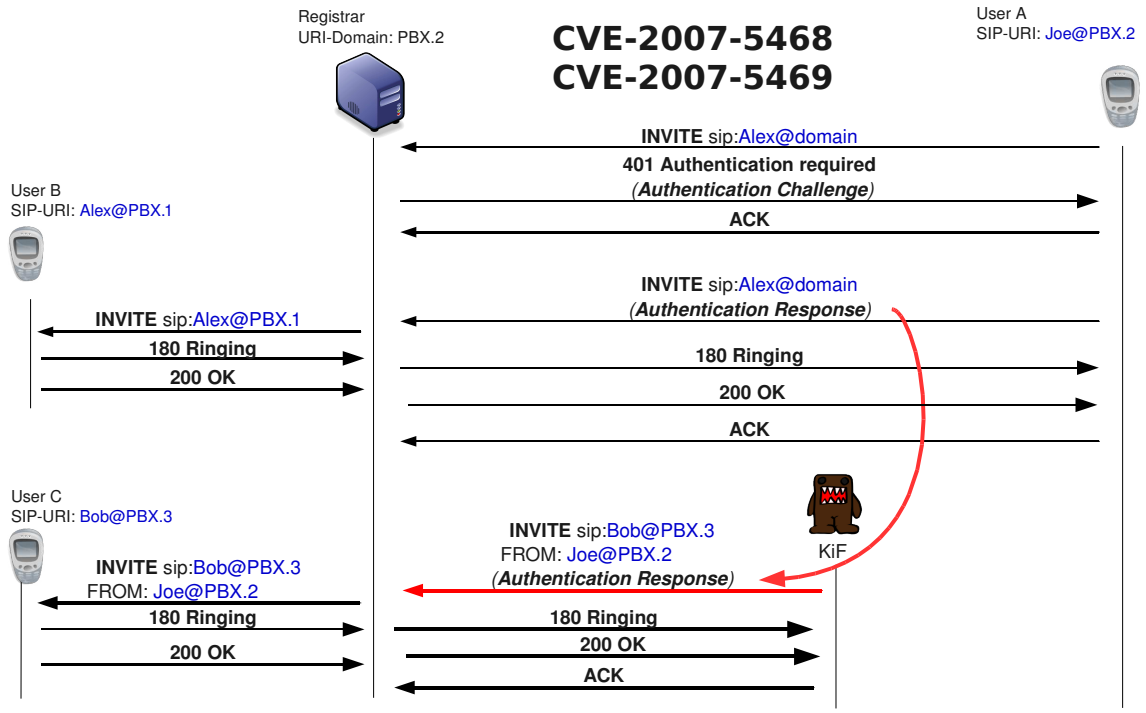


Figure 9.4: Replay attack

the signaling plane can be still tampered with and manipulated by a malicious user. What did change however, is the needed technology. Nowadays, SQL injection commands can be performed in the signaling plane, and the toll fraud is possible.

Some SIP Registrars/Proxies store information gathered from SIP headers into databases. This is necessary for authentication, billing and accounting purposes. If this information is not properly filtered, a SQL injection can be performed. Even more, once it will the information is displayed to the administrator, the data can be interpreted as Java Scripts allowing a XSS attack.

In this case, two consequences can result: First, the database can be changed -for instance the call length can be changed to a small value - and thus the caller can do toll fraud. Second, the administrator can be exposed to an XSS attack (same as described in section 9.1.3).

Asterisk[46] is a popular, largely deployed and open source Linux based VoIP PBX. This software allows to log the Call Detail Records (CDR) in the MySQL database. FreePBX[47] and Trixbox[48] use the information stored in such database in order to manage, compute generate billing reports or display the load of the PBX.

Certain functions do not properly escaped input characters from fields of incoming calls before storing them in the database.

This specific attack may be performed by an user not subscribed in the domain and negative numbers can be inserted in the CDR table in order to change the recorded length/other parameters of a given call. The direct consequence is that no accurate accounting is performed and the charging process is completely controlled by an attacker.

A second and more serious consequence is that this attack can be escalated by injecting

[46] `http://www.asterisk.org/` last checked on December 2008
[47] `http://freepbx.org` last checked on December 2008
[48] `http://www.trixbox.com/` last checked on December 2008

JavaScript [125] tags to be executed by the administrator PC when he/she will perform simple management operations. In this case, a Cross-Site Scripting Attack (XSS) [125] attack is resulted, because malicious JavaScript can be stored into the database by the SQL injection. This malware gets executed on the browser when the administrator will check it - this is a very similar process to the log injection attacks known by the Web application security community. Similarly to the previous case, tools like Beef and XSS proxy can scan the internal network,deactivate firewalls and realize all the CSRF/XSRF specific attacks

The main issue is that most current applications that deal with CDR data are not considering this type of threat. If the target system is not well secured, SQL injection can lead to system compromise because most database server allow some interaction with the target OS [95].

This type of vulnerability is rather dangerous because few application implement filtering on SIP headers. All applications do consider SIP related information to be sourced from a trusted origin and no security screening is performed. The mitigation should be proper input and output filtering whenever data is stored/read from another software component.

9.2 Protocol Design Flaw

Our main work consisted in searching for vulnerabilities in specific SIP implementations without considering the security of the SIP protocol itself. We were however surprised to discover during a complex fuzzing scenario the same anomaly (and apparent vulnerability) shared by all devices under test. Under a more careful analysis, we did realize that in fact the vulnerability comes from the SIP protocol itself and therefore makes toll fraud possible on any VoIP network [49]. The major issue is that this attack, called Relay Attack, is possible by forcing a called party to issue a re-Invite operation.

Due to the novelty and severity of it, section 9.3 describes the known Replay Attack which will allow to understand the difference with the Relay Attack which is described in details at section 9.4. A formal validation using AVISPA [1] have been performed in cooperation with the CASSIS research team of INRIA [7].

9.3 Replay Attack

When SIP is deployed without any underling cryptographic protection mechanism, the typical man in the middle and impersonation attacks between a caller and its proxy, (see Figure 9.4) are straightforward. However, these must are constrained by some important factors . Firstly, the attacker willing to impersonate the user has to be in the middle of the session path and be able to manipulate the session traffic. Secondly, the attacker cannot trigger the user to make such a call at a specific time. Finally, the attacker is restricted to use the generated response just to call the entity for which the user directed the call. In other words, the attacker is not able to call an entity of its choice.

However, if the nonces are not correctly check to be one time used, the third argument could be bypassed since

> *"... in SIP, a server MAY check that the Request-URI in the Authorization header field value corresponds to a user for whom the server is willing to accept forwarded or direct requests, but it is not necessarily a failure if the two fields are not equivalent."*
>
> SIP RFC [120]

[49] `http://voipsa.org/pipermail/voipsec_voipsa.org/2007-November/002475.html` last checked on December 2008

9.4 Design Flaw (Relay Attack)

The synopsis is as follow: an attacker will issue a call directly to the victim, the victim answers and later on, puts the attacker on hold (transfers him to any other place or uses any other method which requires a re-INVITE). Once the attacker receives the re-INVITE specifying the "On hold", he will immediately request the victim to authenticate. This last authentication may be used by the attacker to impersonate the victim at its own proxy.

Note, that to perform this attack, there are two headers in the INVITE message that are essential. The **Contact** header has to have the destination call that the attacker wants to call, because, as specified by SIP [120], this information will be used to generate the message by the user entity. The **Record-Route** header specifies that all outgoing messages from the user entity go directly to that entity.

Notations:

- P is the proxy located at URL: proxy.org
- X is the attacker located at URL: attacker.lan.org
- V is the victim located at URL: victim.lan.org
- V is also registered with P under the username victim at proxy.org
- Y is the accomplice of X (it can be in fact X), but we use another notation for clarity sake

The described attack will show how X calls a toll fraud number 1-900-XXXX impersonating V.

1. X calls's directly V.
   "The route set MUST be set to the list of URIs in the Record-Route header field from the request...The remote target MUST be set to the URI from the Contact header field of the request." RFC 3261 [120] Section 12.1.1 UAS behaviour

```
X ---------- INVITE victim.lan.org -------------> V
     From : attacker at attacker.lan.org
     To: victim at victim.lan.org
     Contact: 1900-XXXX at proxy.org
     Record-Route: attacker.lan.org
```

2. The normal SIP processing

```
X <--------------- 180 Ringing ------------------ V
X <----------------- 200 OK ---------------------- V
X <--------------- Media Data ------------------> V
```

3. The accomplice Y steps in and invites victim V, and then the victim decides to put X on hold

4. The victim, V, sends a re-INVITE to X (to put it on hold)
   "The UAC uses the remote target and route set to build the Request-URI and Route header field of the request." RFC 3261 [120] 12.2.1.1 Generating the Request (Requests within a Dialog)

```
X <----------- INVITE 190XXXX at proxy.org ------ V
     From: victim at victim.lan.org
     To : attacker at attacker.lan.org
```

5. X calls 1900-XXXX using the proxy P and the proxies asks X to authenticate using a Digest Access Authentication with nonce="Proxy-Nonce-T1" and realm ="proxy.org"

6. X request the victim to authenticate the re-INVITE from step 4 using the same Digest Access Authentication received in step 5

```
X ------------401/407 Authenticate -------------> V
      Digest: realm ="proxy.org",
              nonce="Proxy-Nonce-T1"
```

7. In this step the victim will do the work for X (Relay Attack)

```
X <----------- INVITE 190XXXX at proxy.org -------- V
      Digest: realm ="proxy.org",
              nonce="Proxy-Nonce-T1"
              username= "victim",
              uri="1900XXXX at proxy.org",
              response="the victim computed response"
```

8. X may reply now to the Proxy with the valid Digest Access Authentication computed by the victim. Note that the Digest itself it is a perfectly valid one.

Figure 9.5 summarized the whole attack.

9.5 Mitigation

Authentication challenges in SIP are computed using pieces of information extracted from the authenticate message plus the username and shared secret. In the simplest case the authentication response is computed by:

```
A1    = username ":" realm ":" passwd
A2    = Method ":" Digest-URI
response = MD5(MD5(A1) ":" nonce ":" MD5(A2))
```

where **response** is the actual authentication response (as explained in Chapter 1.1.5). Thus, the computed authentication responses will be rejected if the method of the message is different than the method used to generate the response.

However, the described attack abuses that restriction due to the fact that SIP defines an INVITE method which can be used in different contexts (i.e. for initiation of a session and renegotiation). Therefore, the variable $A2$ is the same in both contexts. If different methods names are used for those contexts, then the generated authentication response cannot be used for such an attack.

We propose a mitigation that consists in defining the re-INVITE method as a proper method with a new name: RE-INVITE. Note that computed authentication for such message will use the RE-INVITE method in the variable $A2$ rather than INVITE. Thus, it will generate an authentication token useful only for re-INVITEs messages. Our proposed solution is simple and it should not require to much modifications in the overall protocol.

9.6 Conclusion

Our conclusions after a long term work on searching vulnerabilities in the VoIP space are rather pessimistic. Feedback and support when contacting vendors remains highly unpredictable and

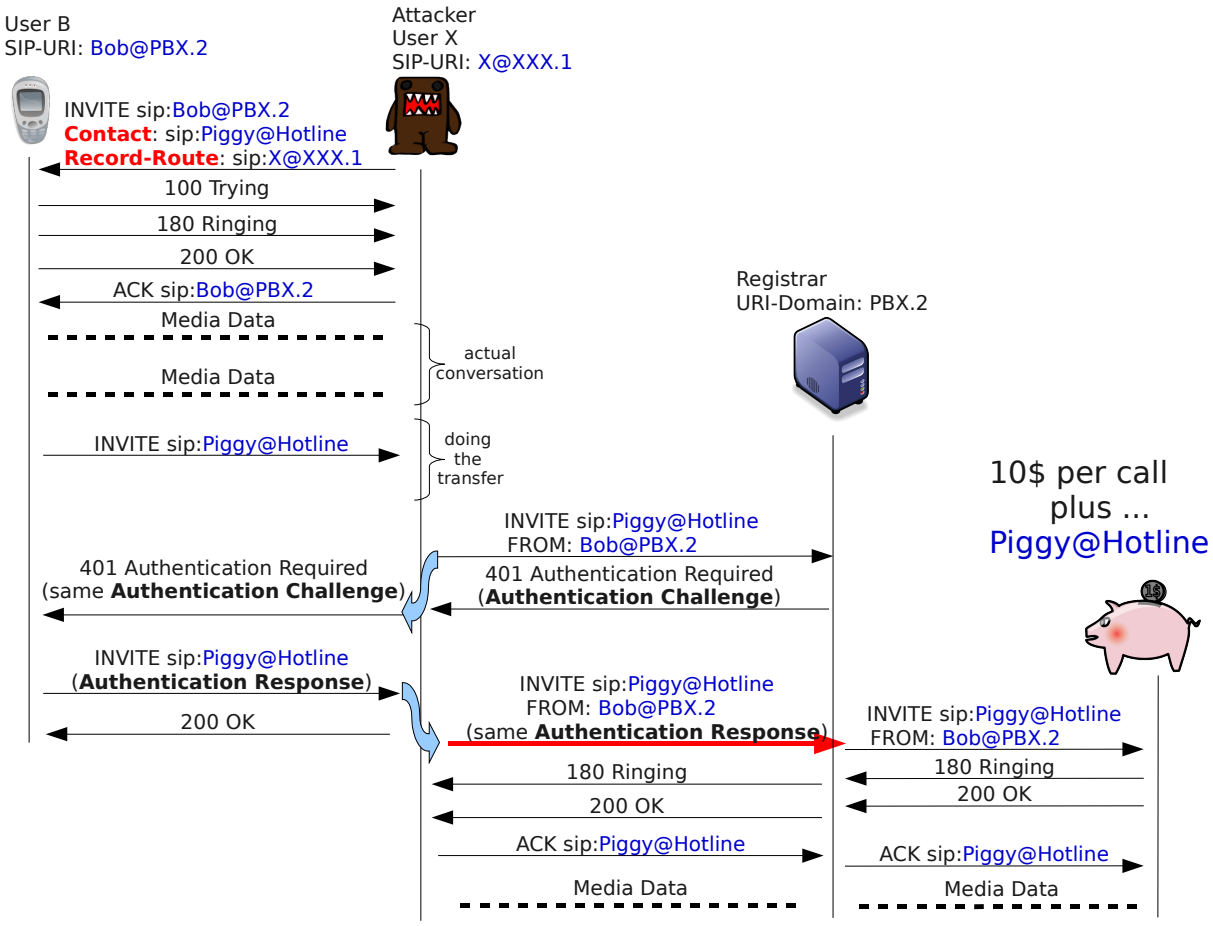


Figure 9.5: Relay attack to SIP summarized

poor. All tested devices have been found vulnerable. The scope of the detected vulnerabilities is very large. Trivial input validation vulnerabilities affecting highly sensitive communication materials are rather usual. More complex and protocol tracking related ones do also exist, though their discovery and exploitation is rather complex. The cause of these vulnerabilities is the weak software security life-cycle of their vendors. The integration of Web and VoIP technology is a Pandora's box comprising even more powerful and hidden dangers. Web specific attacks can be carried out over the SIP plane leading to potential devastating effects, like for instance the complete compromise of an internal network. This is possible since no application specific firewall today can easily interact with several technologies and no proper guidelines for the secure Web and VoIP exist. The more structural cause is a missing VoIP specific threat model. The VOIPSA did develop a threat model [58] which however does not reflect the current state. Highly efficient Denial of Service attacks can be done with single-shot packets, remote eavesdropping goes beyond the simple call interception and the VoIP plane itself can be a major security threat to the overall IT infrastructure. Much remains to be done in the future, among which "Security Build in VoIP devices" remains the major among them. Changes in the software development cycles must be followed by an comprehensive security assessment and testing. Protocol fuzzing is one essential building block in this landscape, since no other additional approach can be used by independent security research. We have described in this chapter our practical and hand-on experience in testing embedded SIP stack implementation. These tests were performed in order to validate our

research on advanced security fuzzing techniques and the discovered vulnerabilities were properly and responsibly disclosed.

Conclusions

Our research performed in the MADYNES research group, addresses the secure management of VoIP networks. We work towards an integrated system able to retrieve as much as possible information from the target environment. Such data can be used by assessment tests and can build scenarios allowing to identify where the flaws of the system are located. This thesis mainly focuses on VoIP networks but our approaches as well as the designed architectural model can be extended to other types of networks or services.

This thesis shows how accurate information from the network can be gathered, and provides an Information Model capable to represent it in an appropriate way for assessment methods. A network information model capable to represent the information required to perform VoIP assessment is part of the proposed model. A VoIP assessment architecture and its implementation has been described. A framework based on attack tree modeling in order to represent and write VoIP attacks has been built.

A novel approach for generating fingerprinting systems based on the structural analysis of protocol messages has been designed. Most existing fingerprinting systems are built manually and require a long lasting development process. Our solution automates the discrimination of signatures by using a structural approach, where formal grammars and collected network traffic are used. It detects important and relevant complex tree like structures and leverages them for building fingerprints.

The applicability of our solution lies in the field of intrusion detection and security assessment, where precise device/service/stack identification are essential. A SIP specific fingerprinting system has been implemented and its performance has been evaluated. The obtained results are very encouraging. This is due to the fact that a structural message analysis is performed. Future work will consist in improving the method and applying it to other protocols and services and towards the natural evolution, where the underlying grammar is unknown. Features are identified by paths and their associated values in the parse tree.

A stateful protocol fuzzer for SIP has been described. The main contribution is a flexible, adaptive fuzzer capable to track the state of the targeted application and device. One of the components of this work is quite generic and reusable for any protocol for which an underlying grammar is known. The second one is dependent on the domain specifics (SIP). To the best of our knowledge, this is the first SIP fuzzer capable to go beyond the simple generation of random input data. The quantitative conclusions after a long term work on searching vulnerabilities in the VoIP space are surprisingly high. Feedback and support when contacting vendors remains highly unpredictable and poor. All tested devices have been found vulnerable. The scope of the detected vulnerabilities is very large. Trivial input validation vulnerabilities affecting highly sensitive communication materials are rather usual. More complex and protocol tracking related ones do also exist, though their discovery and exploitation is rather complex. The cause of these vulnerabilities is the weak software security life-cycle of their developers. The integration of Web and VoIP technology is a Pandora's box comprising even more powerful and hidden dangers. Web specific attacks can be carried out over the SIP plane leading to potential devastating

effects, like for instance the complete compromise of an internal network. This is possible since no application specific firewall today can easily interwork with several technologies and no proper guidelines for the secure interworking of Web and VoIP exist. The more structural cause is a missing VoIP specific threat model. The VoIPSA alliance did develop a threat model which does not fully reflect the current state. Highly efficient Denial of Service attacks can be done with single-shot packets, remote eavesdropping goes beyond the simple call interception and the VoIP plane itself can be a major security threat to the overall IT infrastructure. Much remains to be done in the future, among which "Security Build in VoIP devices" remains the major among them. Changes in the software development cycles must be followed by an comprehensive security assessment and testing. Protocol fuzzing is one essential building block in this landscape, since no other additional approach can be used by independent security research. We have described our practical and hand-on experience in testing embedded SIP stack implementation. These tests were performed in order to validate our research on advanced security fuzzing techniques and the discovered vulnerabilities were properly and responsibly disclosed.

The work presented in this manuscript has been published in parts in international conferences and magazines. The work of security assessment presented in chapter 5 has been published in IM 2007[50] [9]. The content of chapter 6 related to passive fingerprinting has been published in RAID 2008[51] [11] and a live demonstration has been conducted at IPTComm 2008[52]. Chapter 8 describing a stateful SIP fuzzer has been published at IPTComm 2007[53] [10]. A public demonstration of the tool has taken place in ShmooCon 2008[54] [14]. The fuzzing experiences presented in chapter 9 have been published in EICAR 08[55] [13] and in MISC #39[56] [12].

The resulting tool of our fuzzing approach, KiF, is open source and available free of charge at `http://kif.gforge.inria.fr/` under an special license to avoid malicious use. Currently, the tool is been partially supported by a cooperation lab between Alcatel-Lucent and INRIA.

Future Work

Assessment related tasks are usually more effective when more information is acquired from the network. One useful case would be to known the common rules or patterns used by firewalls. Thus, their inspection can speed up the process of assessment techniques for the search of vulnerabilities. Our current activities consist in how such rules can be used for improving assessment to find the rules weaknesses and to possibly reforce them.

Currently a standardisation proposal for a common log file for SIP has been submitted to the 74^{th} Internet Engineering Task Force (IETF). The standardisation of the log format can be a valuable work since it will allow different researchers to evaluate their own approaches across different traces, even more for IDS which can detect abnormal behavior in real time.

The fingerprinting approach was limited to study the structural representation of the message, however we realized that there are fields which have some relationship within each other. An interesting study case can be directly related to measure the entropy between fields. This entropy can be analyzed between messages of one session or even between fields of an individual message. Other concerns are related to actively fingerprinting devices and automatically

[50] IM 2007: the 10^{th} IFIP/IEEE Symposium on Integrated Management

[51] RAID 2008: the 11^{th} international symposium on Recent Advances in Intrusion Detection, Boston, USA

[52] IPTComm '08: the 2^{nd} international conference on Principles, Systems and Applications of IP Telecommunications, Heidelberg, Germany

[53] IPTComm '07: the 1^{st} international conference on Principles, Systems and Applications of IP Telecommunications, New York, USA

[54] ShmooCon 2008, Washington DC, USA

[55] EICAR 08: the 17^{th} Annual Conference of the European Institute for Computer Anti-Virus Research, Laval, France

[56] The MISC Magazine - Edition française: Multi-System & Internet Security Cookbook

identifying meaningful queries using our fuzzing approach.

In the short term, we will focus on extending our fuzzing framework by addressing additional protocols, case studies and implementations. Surface covering is an active field for fuzzer comparison. Our current research is oriented towards comparison made out of virtualization environment where the full behavior of the system can be observed and analyzed. An immediate follow up goal is where feedback gathered from the assessed platform can be used to evaluate and drive the fuzzing process. It is important during a fuzzing process to cover as much as possible of executed code in a target application and be able to fine-tune and learn when and how to shift the fuzzing process. It is also essential to know, in a fuzzing scenario, when to stop because no other step can provide any valuable advance.

One definition of performance in the fuzzing landscape is related to the capability to adapt to an unknown protocol. Sometimes, the specifications of a protocol are not available, or too complex in order to take them into account when designing a specific fuzzer. Future work will address this task by coupling work on automated reverse engineering of unknown protocols and fuzzing techniques. The first goal is to develop and validate sound approaches for reverse engineering of an unknown protocol. The best illustration is the following: starting from a network capture, can the individual protocol message units and associated message structure by automatically identified? The second sub-goal is to derive a fuzzing framework where the discovered protocol elements can be automatically fuzzed.

Bibliography

[1] Avispa project. http://www.avispa-project.org.

[2] CDP: Cisco Discovery Protocol. http://www.cisco.com/univercd/cc/td/doc/product/lan/trsrb/frames.htm#xtocid12.

[3] Signalling System #7. SS7 International Telecommunication Union (ITU-T), 1980.

[4] Common Information Model (CIM). Distributed Management Task Force, Inc., DSP 0004 (Standard), 1999.

[5] VoIP Availability and Reliability Model for PacketCable Architecture. PacketCable, http://www.searchuu.com/show/60684-VoIP-Availability-and-Reliability-Model-for-the-PacketCable-..., Nov. 2000.

[6] Voice Over Internet Protocol (VOIP) Security Technical Implementation Guide. Defense Information Systems Agency, http://csrc.nist.gov/pcig/STIGs/VoIP-STIG-V1R1R-4PDF.pdf, Jan. 2004.

[7] H. J. Abdelnur, T. Avanesov, M. Rusinowitch, and R. State. Abusing SIP Authentication. In IAS '08: Proceedings of the 2008 The Fourth International Conference on Information Assurance and Security, pages 237–242, Washington, USA, 2008. IEEE Computer Society.

[8] H. J. Abdelnur, V. Cridlig, R. State, and O. Festor. VoIP Security Assessment: Methods and Tools. In 1st IEEE Workshop on VoIP Management and Security (VoIP MaSe 2006), pages 29–34, Vancouver, Canada, Apr. 2006.

[9] H. J. Abdelnur, R. State, I. Chrisment, and C. Popi. Assessing the security of VoIP Services. In The 10th IFIP/IEEE Symposium on Integrated Management (IM 2007), Munich, Germany, May 2007.

[10] H. J. Abdelnur, R. State, and O. Festor. KiF: a stateful SIP fuzzer. In IPTComm '07: Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications, pages 47–56, New York, USA, 2007. ACM.

[11] H. J. Abdelnur, R. State, and O. Festor. Advanced Network Fingerprinting. In RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, pages 372–389, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] H. J. Abdelnur, R. State, and O. Festor. Failles et VoIP. In MISC Magazine - Edition française: Multi-System & Internet Security Cookbook. Misc #39, Septembre/Octobre 2008.

[13] H. J. Abdelnur, R. State, and O. Festor. Fuzzing for vulnerabilities in the VoIP space. In EICAR '08: 17th Annual Conference of the European Institute for Computer Anti-Virus Research, Laval France, 2008.

[14] H. J. Abdelnur, R. State, and O. Festor. SIPping your Network. In Shmoocon 2008, Washington, USA, Feb. 2008.

[15] C. Adams, S. Farrell, T. Kause, and T. Mononen. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Proposed Standard), Sept. 2005.

[16] M. Ahmed, E. Al-Shaer, and L. Khan. A Novel Quantitative Approach For Measuring Network Security. INFOCOM 2008. The 27th Conference on Computer Communications. IEEE, pages 1957–1965, Apr. 2008.

[17] D. Aitel. The Advantages of Block-Based Protocol Analysis for Security Testing. Immunity Inc, `http://www.immunitysec.com/resources-papers.shtml`, Feb. 2002.

[18] E. Al-Shaer, L. Khan, and M. S. Ahmed. A comprehensive objective network security metric framework for proactive security configuration. In CSIIRW '08: Proceedings of the 4th annual workshop on Cyber security and informaiton intelligence research, pages 1–3, New York, USA, 2008. ACM.

[19] O. Arkin. Demystifying Passive Network Discovery and Monitoring Systems. `http://www.usenix.org/publications/login/2005-06/pdfs/arkin0506.pdf`, June 2005. The USENIX Magazine.

[20] M. Balaban. Buffer Overflows Demystified. `http://www.enderunix.org/documents/eng/bof-eng.txt`, 2001.

[21] R. Baldwin. Rule Based Analysis of Computer Security. Technical report, Cambridge, MA, USA, 1987.

[22] G. Banks, M. Cova, V. Felmetsger, K. C. Almeroth, R. A. Kemmerer, and G. Vigna. SNOOZE: Toward a Stateful NetwOrk prOtocol fuzZEr. In of Lecture Notes in Computer Science, pages 343–358. Springer, 2006.

[23] E. Bayse, A. R. Cavalli, M. Núñez, and F. Zaïdi. A passive testing approach based on invariants: application to the WAP. Computer Networks, 48(2):235–245, 2005.

[24] M. Beddoe. The Protocol Informatics Project. In Toorcon, Sept. 2004. San Diego, USA.

[25] B. Beizer. Software Testing Techniques. Van Nostrand Reinhold Company, 1st edition, 1982.

[26] S. M. Bellovin. A technique for counting natted hosts. In IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, pages 267–272, New York, USA, 2002. ACM.

[27] T. Berg, B. Jonsson, and H. Raffelt. Regular Inference for State Machines with Parameters. In L. Baresi and R. Heckel, editors, FASE, volume 3922 of Lecture Notes in Computer Science, pages 107–121. Springer, 2006.

[28] L. Bernaille. Classification temps réel d'applications sur l'Internet. PhD thesis, Université Pierre et Marie Curie, 2007.

[29] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference, pages 1–12, New York, USA, 2006. ACM.

[30] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. SIGCOMM Comput. Commun. Rev., 37(4):37–48, 2007.

[31] N. Borisov, D. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo. A Generic Application-Level Protocol Analyzer and its Language, 14h Symposium on Network and Distributed System Security . In 14th Symposium on Network and Distributed System Security, 2007.

[32] T. Bowen, J. Haluska, P. Thermos, and S. Ungar. Performance and Security Analysis of SIP using IPSec. In VoIP Security - Challenges and Solutions, GlobeCom 2004 Workshop, Dallas, USA, Dec. 2004.

[33] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), Oct. 1989.

[34] A. Broder. On the Resemblance and Containment of Documents. In SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997, page 21, Washington, USA, 1997. IEEE Computer Society.

[35] D. Brumley, J. Caballero, Z. Liang, J. Newsome, and D. Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pages 1–16, Berkeley, USA, 2007. USENIX Association.

[36] D. Buttler. A Short Survey of Document Structure Similarity Algorithms. In The 5th International Conference on Internet Computing, June 2005.

[37] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum. FiG: Automatic Fingerprint Generation. In The 14th Annual Network & Distributed System Security Conference (NDSS 2007), Feb. 2007.

[38] C. Cadar, P. Twohey, V. Ganesh, and D. Engler. EXE: A System for Automatically Generating Inputs of Death Using Symbolic Execution. In In Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS), Virginia, USA, Nov. 2006.

[39] M. Chang and C. K. Poon. Catching the Picospams. In In International Syposium on Methodologies for Intelligent Systems (ISMIS 2005), pages 641–649, New York, USA, May 2005. Springer.

[40] E. J. Chikofsky and J. H. C. II. Reverse Engineering and Design Recovery: A Taxonomy. Software, IEEE, 7(1):13–17, Jan. 1990.

[41] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234 (Standard), Jan. 2008.

[42] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. SIGCOMM Comput. Commun. Rev., 37(1):5–16, 2007.

[43] W. Cui, J. Kannan, and H. J. Wang. Discoverer: automatic protocol reverse engineering from network traces. In SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pages 1–14, Berkeley, USA, 2007. USENIX Association.

[44] W. Cui, V. Paxson, N. Weaver, and R. H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In NDSS. The Internet Society, 2006.

[45] M. Dacier and Y. Deswarte. Privilege Graph: an Extension to the Typed Access Matrix Model. In ESORICS '94: Proceedings of the Third European Symposium on Research in Computer Security, pages 319–334, London, UK, 1994. Springer-Verlag.

[46] M. Dacier, Y. Deswarte, and M. Kaâniche. Models and tools for quantitative assessment of operational security. pages 177–186, 1996.

[47] O. Dahl and S. Wolthusen. Modeling and Execution of Complex Attack Scenarios using Interval Timed Colored Petri Nets. In IWIA '06: Proceedings of the Fourth IEEE International Workshop on Information Assurance (IWIA'06), pages 157–168, Washington, USA, 2006. IEEE Computer Society.

[48] G. Dalton, R. Mills, J. Colombi, and R. Raines. Analyzing Attack Trees using Generalized Stochastic Petri Nets. Information Assurance Workshop, 2006 IEEE, pages 116–123, June 2006.

[49] J. Davidson, B. Gracely, and J. Peters. Overview of the PSTN and Comparisons to Voice over IP. Cisco Press, Jan. 2001.

[50] J. DeMott. The Evolving Art of Fuzzing. In Defcon 14, Las Vegas, USA, Aug. 2006.

[51] J. Demott, R. Enbody, and W. .Punch. Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing. In Black Hat 2007, Las Vegas, USA, Aug. 2007.

[52] D. E. Denning. An Intrusion-Detection Model. IEEE Trans. Softw. Eng., 13(2):222–232, 1987.

[53] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008.

[54] Douglas Comer and John C. Lin. Probing TCP implementations. In USENIX Summer, pages 245–255, 1994.

[55] W. Drewry and T. Ormandy. Flayer: exposing application internals. In WOOT '07: Proceedings of the first USENIX workshop on Offensive Technologies, pages 1–9, Berkeley, USA, 2007. USENIX Association.

[56] L. Dryburgh and J. Hewett. Signaling System No. 7 (SS7/C7): Protocol, Architecture, and Applications. Cisco Press, 2003.

[57] S. Embleton, S. Sparks, and R. Cunningham. Sidewinder: An Evolutionary Guidance System for Malicious Input Crafting. In Black Hat 2007, Las Vegas, USA, Aug. 2006.

[58] D. Endler, D. Ghosal, R. Jafari, A. K. M. Kolenko, N. Nguyen, W. Walkoe, and J. Zar. VoIP Security and Privacy Threat Taxonomy. VOIPSA. `http://voipsa.org/Activities/taxonomy.php`, Oct. 2005.

[59] D. Farmer and E. Spafford. The COPS Security Checker System. In USENIX Summer, pages 165–170, 1990.

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999.

[61] O. Filols and A. Rüegg. The security bug catcher. `http://lasecwww.epfl.ch/~oechslin/projects/bugcatcher/`, 2004.

[62] S. Flesca, G. Manco, E. Masciari, and L. Pontieri. Fast Detection of XML Structural Similarity. IEEE Transactions on Knowledge and Data Engineering, 17(2):160–175, 2005.

[63] B. Ford. Parsing expression grammars: a recognition-based syntactic foundation. In POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 111–122, New York, USA, 2004. ACM.

[64] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for Unix processes. pages 120–128, May 1996.

[65] M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In QoP '08: Proceedings of the 4th ACM workshop on Quality of protection, pages 23–30, New York, USA, 2008. ACM.

[66] A. Fritzler. UnOfficial AIM/OSCAR Protocol Specification. `http://www.oilcan.org/oscar/`, 2007.

[67] Open Source FastTrack P2P Protocol. `http://gift-fasttrack.berlios.de/`, 2007.

[68] P. Godefroid, A. Kieźun, and M. Y. Levin. Grammar-based Whitebox Fuzzing. In PLDI'2008: ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, US, 2008.

[69] P. Godefroid, M. Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), pages 151–166, San Diego, USA, Feb. 2008.

[70] K. Gopalratnam, S. Basu, J. Dunagan, and H. J. Wang. Automatically Extracting Fields from Unknown Network Protocols. In Systems and Machine Learning Workshop 2006, Saint-Malo, France, June 2006.

[71] S. Granger. Social Engineering Fundamentals. Security Focus, `http://www.securityfocus.com/infocus/1527`, Dec. 2001.

[72] S. Gupta and J. Winstead. Using Attack Graphs to Design Systems. Security & Privacy, IEEE, 5(4):80–83, July 2007.

[73] H. Scholz. SIP Stack Fingerprinting and Stack Difference Attacks. In Black Hat 2007, Las Vegas, USA, July 2006.

[74] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. Acas: automated construction of application signatures. In MineNet '05: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data, pages 197–202, New York, USA, 2005. ACM.

[75] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[76] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. In SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium, pages 9–9, Berkeley, CA, USA, 2001. USENIX Association.

[77] I. G. Harris. INTERSTATE: A Stateful Protocol Fuzzer. In Defcon 15, Las Vegas, USA, Aug. 2007.

[78] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Uncover Security Design Flaws Using The STRIDE Approach. Microsoft Corporation. `http://msdn.microsoft.com/en-us/library/ms954176.aspx`, 2006.

[79] M. Howard and S. Lipner. Inside the Windows security push. Security & Privacy, IEEE, 1(1):57–61, Jan.-Feb. 2003.

[80] Y. Hsu, G. Shu, and D. Lee. A model-based approach to security flaw detection of network protocol implementations. Network Protocols, 2008. ICNP 2008. IEEE International Conference on, pages 114–123, Oct. 2008.

[81] D. Ignjatic, L. Dondeti, F. Audet, and P. Lin. MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY). RFC 4738 (Proposed Standard), Nov. 2006.

[82] C. Jennings and N. Modadugu. Session Initiation Protocol (SIP) over Datagram Transport Layer Security. draft-jennings-sip-dtls-05 (Internet Draft), Apr. 2008.

[83] R. Kaksonen. A Functional Method for Assessing Protocol. Implementation Security. VTT Electronics. VTT Publications 448. 128 p. + app. 15 p, 2001.

[84] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. SIGCOMM Comput. Commun. Rev., 35(4):229–240, 2005.

[85] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005.

[86] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. Dependable and Secure Computing, IEEE Transactions on, 2(2):93–108, Apr. 2005.

[87] D. R. Kuhn, T. J. Walsh, and S. Fries. Security Considerations for Voice Over IP Systems. National Institute of Standards and Technology, `http://csrc.nist.gov/publications/nistpubs/800-58/SP800-58-final.pdf`, 2005.

[88] A. S. L. Wang and S. Jajodia. Measuring the overall security of network configurations using attack graphs. In Proceedings of 21th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'07), 2007.

[89] H. Lau. MPack, Packed Full of Badness. `http://www.symantec.com/enterprise/security_response/weblog/2007/05/mpack_packed_full_of_badness.html`, June 2007.

[90] D. Lee, D. Chen, R. Hao, R. Miller, J. Wu, and X. Yin. A Formal Approach for Passive Testing of Protocol Data Portions. In ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols, pages 122–131, Paris, France, 2002. IEEE Computer Society.

[91] C. Leita, K. Mermoud, and M. Dacier. ScriptGen: an automated script generation tool for honeyd. In ACSA 2005, 21st Annual Computer Security Applications Conference, Dec. 2005.

[92] K.-S. Lhee and S. J. Chapin. Buffer overflow and format string overflow vulnerabilities. Software Practical Experiences, 33(5):423–460, 2003.

[93] L. Li, N. Krasnogor, and J. Garibaldi. Automated self-assembly programming paradigm: Initial investigations. In The Third IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, pages 25–34, Potsdamn, Germany, 2006. IEEE Computer Society.

[94] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic Protocol Format Reverse Engineering through Conectect-Aware Monitored Execution. In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), San Diego, USA, Feb. 2008.

[95] D. Litchfield, C. Anley, J. Heasman, and B. Grindlay. The Database Hacker's Handbook: Defending Database Servers. John Wiley & Sons, 2005.

[96] G. Lyon. Remote OS Detection via TCP/IP Fingerprinting (2nd Generation). `http://nmap.org/book/osdetect.html`.

[97] G. Lyon. The Art of Port Scanning. `http://nmap.org/p51-11.html`.

[98] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pages 313–326, New York, USA, 2006. ACM.

[99] G. Malan, D. Watson, F. Jahanian, and P. Howell. Transport and application protocol scrubbing. In INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1381–1390 vol.3, Mar 2000.

[100] P. M. Maurer. Generating Test Data with Enhanced Context-Free Grammars. IEEE Softw., 7(4):50–55, 1990.

[101] J. P. McDermott. Attack net penetration testing. In NSPW '00: Proceedings of the 2000 workshop on New security paradigms, pages 15–21, New York, USA, 2000. ACM.

[102] C. McNab. Network Security Assessment. O'Reilly; 1 edition, Mar. 2004.

[103] L. Miclet, P. Dupont, and S. Vial. Inférence Grammaticale Régulière : méthodes semi-itératives et mesure de performance. In JFA 95, Journées Francophones sur l'Apprentissage, pages 103–106, Grenoble, France, 1995.

[104] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. Communications of the Association for Computing Machinery, 33(12):32–44, 1990.

[105] C. Miller. How smart is Intelligent Fuzzing - or - How stupid is Dumb Fuzzing? In Defcon 15, Las Vegas, USA, Aug. 2007.

[106] M. Mintz-Habib, A. Rawat, H. Schulzrinne, and X. Wu. A VoIP emergency services architecture and prototype. Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference, pages 523–528, 2005.

[107] J. Networks. VoIP Security - best practices Outline. `http://www.juniper.net/solutions/literature/white_papers/200179.pdf`, 2006.

[108] J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: automatic protocol replay by binary analysis. In CCS '06: Proceedings of the 13th ACM conference on Computer and communications security, pages 311–321, New York, USA, 2006. ACM.

[109] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In ICSE'07, Proceedings of the 29th International Conference on Software Engineering, pages 75–84, Minneapolis, USA, May 2007.

[110] V. Paxson. Automated packet trace analysis of TCP implementations. SIGCOMM Comput. Commun. Rev., 27(4):167–179, 1997.

[111] C. Peikari and A. Chuvakin. Security Warrior. O'Reilly & Associates, Inc., Sebastopol, USA, 2004.

[112] J. L. Peterson. Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[113] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In NSPW '98: Proceedings of the 1998 workshop on New security paradigms, pages 71–79, New York, USA, 1998. ACM.

[114] T. Porter, J. Kanclirz, A. Zmolek, A. Rosela, M. Cross, L. Chaffin, B. Baskin, and C. Shim. Practical VoIP Security. Andrew Williams, 2006.

[115] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004.

[116] J. F. Ransome and J. Rittinghouse. Voice over Internet Protocol (VoIP) Security. Digital Press, Newton, USA, 2004.

[117] N. Rathaus and G. Evron. Open Source Fuzzing Tools. Syngress (August 1, 2007), 1 edition, Aug. 2007.

[118] E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347 (Proposed Standard), Apr. 2006.

[119] R. Rosenbaum. Secrets of the Little Blue Box. Esquire Magazine, 1971.

[120] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002.

[121] I. Rouvellou and G. Hart. Inference of a probabilistic finite state machine from its output. Systems, Man and Cybernetics, IEEE Transactions on, 25(3):424–437, Mar. 1995.

[122] B. Schneier. Attack Trees, Modeling Security Threats. Dr. Dobb's Journal, Dec. 1999.

[123] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[124] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia. Voip intrusion detection through interacting protocol state machines. In DSN '06: Proceedings of the International Conference on Dependable Systems and Networks, pages 393–402, Washington, DC, USA, 2006. IEEE Computer Society.

[125] R. H. A. R. P. D. P. Seth Fogie, Jeremiah Grossman. XSS Exploits: Cross Site Scripting Attacks and Defense. Syngress, 2007.

[126] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated Generation and Analysis of Attack Graphs. In SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy, page 273, Washington, USA, 2002. IEEE Computer Society.

[127] G. Shu and D. Lee. Network Protocol System Fingerprinting - A Formal Approach. INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, pages 1–12, Apr. 2006.

[128] A. Slagell, R. Bonilla, and W. Yurcik. A survey of PKI components and scalability issues. Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International, pages 10 pp.–, Apr. 2006.

[129] M. Smart, R. Malan, and F. Jahanian. Defeating TCP/IP stack fingerprinting. In SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium, pages 17–17, Berkeley, CA, USA, 2000. USENIX Association.

[130] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), Sept. 2007.

[131] M. Sutton, A. Greene, and P. Amini. Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley Professional; 1 edition (July 9, 2007), 1 edition, July 2007.

[132] A. Takanen, J. DeMott, and C. Miller. Fuzzing for Software Security Testing and Quality Assurance. Artech House, Inc., Norwood, USA, 2008.

[133] P. Thermos and A. Takanen. Securing VoIP Networks: Threats, Vulnerabilities, and Countermeasures. Addison-Wesley Professional, 2007.

[134] A. Tridgell. How Samba was written. `http://samba.org/ftp/tridge/misc/french_cafe.txt`, 2003.

[135] O. University. PROTOS Test-Suite: c07-sip. `http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip`, 2005.

[136] I. van Sprundel. Fuzzing: Breaking software in an automated fashion. Berlin, Germany, Dec. 2005.

[137] J. M. Voas and G. McGraw. Software fault injection: inoculating programs against errors. John Wiley & Sons, Inc., New York, USA, 1997.

[138] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An Attack Graph-Based Probabilistic Security Metric. In Proceedings of 22th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08), pages 283–296, 2008.

[139] D. Watson, M. Smart, G. R. Malan, and F. Jahanian. Protocol scrubbing: network security through transparent flow modification. IEEE/ACM Trans. Netw., 12(2):261–273, 2004.

[140] H. A. Watson. Launch control safety study. Technical report, Bell Telephone Laboratories, 1961.

[141] D. A. Wheeler. Secure Programming for Linux and Unix HOWTO. `http://www.dwheeler.com/secure-programs`, 2003.

[142] C. Wieser, M. Laakso, and H. Schulzrinne. SIP Robustness Testing for Large-Scale Use. In SOQUA/TECOS, pages 165–178, 2004.

[143] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda. Automatic Network Protocol Analysis. In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), San Diego, USA, Feb. 2008.

[144] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pages 1–12, Berkeley, USA, 2007. USENIX Association.

[145] H. Yan, K. Sripanidkulchai, H. Zhang, Z. Shae, and D. Saha. Incorporating Active Fingerprinting into SPIT Prevention Systems. Third Annual VoIP Security Workshop (VSW'06), June 2006.

[146] P. Zezula, F. Mandreoli, and R. Martoglia. Tree signatures and unordered XML pattern matching. In Proceedings of the 32nd International Conference on Current Trends in Theory and Practice of Computer Science, pages 122–139. Springer, Merin, Czech Republic, 2004.

Glossary

ABNF : Augmented Backus-Naur Form is context-free grammar specification used to formally describe the message syntax in protocol specifications
ARP : Address Resolution Protocol is used for finding a host's MAC address using its IP address

CDP : The Cisco Discovery Protocol is a proprietary protocol used by Cisco devices to advertise themselves and discover other devices in the network
CVE : Common Vulnerabilities and Exposures is a database of publicly known security vulnerabilities and exposures

DHCP : Dynamic Host Configuration Protocol is a protocol used for providing the required information for end points to connect to an IP network
DoS : Denial of Service is a term used when a component is disabled from its normal service
DTLS : Datagram Transport Layer Security protocol provides equivalent protection mechanism to TLS using an unreliable transport layer, like UDP

HTTP : HyperText Transfer Protocol is the protocol for retrieving documents used in the Internet

IDS : Intrusion Detection System is a software designed to detect anomalies, attacks and unwanted manipulation on a network
IETF : Internet Engineering Task Force is an organization which develops and promotes protocol standards for the Internet
IPSec : IP Security allows to create a secure tunnel between end points, thus providing mutual authentication, encryption, anti-replay and data integrity
ISP : Internet Service Provider is a company that provides Internet services

PBX : Private Branch Exchange is a telephone component used by private organizations to carry its telephone calls
PSTN : Public Switched Telephone Network is the traditional public telephony network

RFC : Request For Comments is the document in the IETF used for describing standards
RTP : Real-time Transport Protocol is a protocol used for encapsulating media data in real time

S/MIME : Secure/Multipurpose Internet Mail Extensions is used for message body encryption and it can be used in an end-to-end fashion
SDP : Session Description Protocol is a format used for describing multimedia sessions
SIP : Session Initiation Protocol is a signaling protocol used for establishing, modifing and tearing down multimedia sessions
SIPS : Is a term used for specifing a SIP resource in which a secure connection is required

SQL : Structured Query Language is a language defined for querying and modifying relational databases

SRTP : Secure Real Time Protocol provides authentication, confidentiality, integrity and replay protection of the media data

SS7 : Signaling System #7 is a set of protocols used for managing the communication calls in the PSTN

STRIDE : Is a threat model defined by Microsoft. STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege

TFTP : Trivial File Transport Protocol is used for transfer files. Usually used for network booting

TLS : Transport Layer Security is a cryptographic protocol designed to provide data integrity and confidentiality for TCP traffic between two consecutive hops

UA : User Agent for the SIP protocol (i.e. end point device). It can be logically divided in User Agent Server (UAS) and User Agent Client (UAC) depending on the role in which is playing

UDP : User Datagram Protocol is an unreliable protocol used for simple transmission of data over the network

URI : Uniform Resource Identifier is a string used to identify a resource on the Internet

URL : Uniform Resource Locator specifies the mechanism needed for retrieving a resource on the Internet

VoIP : Voice over IP is a term for the voice communications over the Internet

VoIPSA : Voice over IP Security Alliance (VOIPSA) was therefore created with the purpose of promoting, educating and providing methodologies and tools for people using VoIP services

VPN : Virtual Private Network is a computer network where the nodes do not need to be physically connected instead a tunneled connection is carried on

XML : Extensible Markup Language is a general purpose language used to store or transport data

XSS : Cross-Site Scripting is referred as XSS to not get confused with Cascading Style Sheets. XSS is a type of code injection which allows an attacker to execute malicious code in the client-side of web applications

ZRTP : is defined as a key management scenario for SRTP

Résumé

Les solutions voix sur IP (VoIP) sont actuellement en plein essor et gagnent tous le jours de nouveaux marché en raison de leur faible coût et d'une palette de services riche. Comme la voix sur IP transite par l'Internet ou utilise ses protocoles, elle devient la cible de multiples attaques qui peuvent mettre son usage en péril. Parmis les menaces les plus dangereuses on trouve les bugs et les failles dans les implantations logicielles des équipements qui participent à la livraison de ces services.

Cette thése comprend trois contributions à l'amélioration de la sécurité des logiciels. La première est une architecture d'audit de sécurité pour les services VoIP intégrant découverte, gestion des données et attaques à des fins de test. La seconde contribution consiste en la livraison d'une approche autonome de discrimination de signatures de messages permettant l'automatisation de la fonction de fingerprinting passif utilisée pour identifier de façon unique et non ambigüe la source d'un message. La troisième contribution porte sur la détection dynamique de vulnérabilités dans des états avancés d'une interaction protocolaire avec un équipement cible. L'expérience acquise dans la recherche de vulnérabilités dans le monde de la VoIP avec nos algorithmes est également partagée dans cette thèse.

Mots-clés: Sécurité VoIP, audit de sécurité réssau, fingerprinting passif, extraction de caractéristiques, inférence structurelle de syntaxe, fuzzing, fuzzing de protocoles, vulnérabilités SIP

Abstract

VoIP networks are in a major deployment phase and are becoming widely accepted due to their extended functionality and cost efficiency. Meanwhile, as VoIP traffic is transported over the Internet, it is the target of a range of attacks that can jeopardize its proper functionality. Assuring its security becomes crucial. Among the most dangerous threats to VoIP, failures and bugs in the software implementation will continue rank high on the list of vulnerabilities.

This thesis provides three contributions towards improving software security. The first is a VoIP specific security assessment framework integrated with discovery actions, data management and security attacks allowing to perform VoIP specific assessment tests. The second contribution consists in an automated approach able to discriminate message signatures and build flexible and efficient passive fingerprinting systems able to identify the source entity of messages in the network. The third contribution addresses the issue of detecting vulnerabilities using a stateful fuzzer. It provides an automated attack approach capable to track the state context of a target device and we share essential practical experience gathered over a two years period in searching for vulnerabilities in the VoIP space.

Keywords: VoIP Security, Network Assessment, Passive Network Fingerprinting, Feature extraction, Structural syntax inference, Fuzzing, Protocol Fuzzer, SIP Vulnerabilities