Pycrate : tester les systèmes télécoms et cellulaires avec Python

Benoît Michau
benoit.michau@ssi.gouv.fr

ANSSI

1 Les systèmes télécoms et cellulaires

La plupart des équipements réseaux déployés chez les opérateurs télécoms, mais également les systèmes de télécommunications (modems, antennes-relais...) tels que ceux déployés dans les véhicules (voitures, trains, avions, satellites...) ou autres, sont souvent des systèmes fermés et complexes du point de vue de l'opérateur ou de l'intégrateur : pas de code source disponible, pas de documentation précise du fonctionnement interne, ni d'accès système privilégié (voire pas d'accès système du tout) permettant de superviser facilement les processus internes à l'équipement. Par ailleurs, ceux-ci implémentent la plupart du temps des interfaces répondant à des normes difficiles à appréhender, tout du moins pour une personne novice.

Ainsi, auditer de tels systèmes peut être délicat pour ces raisons, alors même que ceux-ci présentent une exposition importante en terme de sécurité : ces systèmes permettant l'échange de données entre équipements distants, ils sont à la base des réseaux de communications modernes. Afin de pouvoir tester correctement de tels systèmes, il est important de pouvoir travailler efficacement avec les protocoles pris en charge par ceux-ci. Il s'agit souvent de protocoles binaires, s'appuyant parfois sur des syntaxes de description particulières, telles qu'ASN.1 ou CSN.1.

La bibliothèque présentée dans cet article propose de nombreuses fonctionnalités, ainsi que de nombreux formats, facilitant le travail de test et d'évaluation de ces systèmes. Pycrate est disponible en source ouverte sur GitHub : `https://github.com/anssi-fr/pycrate/`

2 Fonctionnalités intégrées

Pycrate est une suite de bibliothèques entièrement écrite en Python, compatible avec les deux versions courantes du langage : Python 2 (à partir de la version 2.7) et Python 3 (à partir de la version 3.4). Elle offre un grand nombre de services lorsqu'on souhaite travailler avec des systèmes cellulaires, ou plus largement de télécommunications :

- `pycrate_core` propose un grand nombres de fonctions, ainsi que la classe `charpy`, pour la conversion de données (principalement entre entiers et *bytes*), sans nécessiter d'alignement à l'octet.
- `pycrate_asn1c` propose un compilateur ASN.1 supportant une très grande partie de la notation ASN.1 et donc de très nombreux modules (par exemple, l'ensemble des protocoles UMTS et LTE ainsi que ceux transportés sur SS7, X.509 et autres standards de PKI, Kerberos, LDAP...).
- Ces modules fonctionnent avec un moteur *(runtime)* d'encodage/décodage disponible dans `pycrate_asn1rt` ; celui-ci supporte les codecs BER, CER, DER, ainsi que PER aligné et non-aligné. Pycrate permet de travailler ainsi avec la plupart des formats utilisant ASN.1.
- `pycrate_csn1` permet de traduire en Python les formats décrits en CSN.1 et de les manipuler aisément. CSN.1 est une notation associée à une méthode d'encodage bit à bit, très utilisée dans les systèmes GSM et GPRS.
- `pycrate_mobile` met à disposition un grand nombre de structures et de fonctions pour manipuler les messages de signalisation utilisés dans les réseaux mobiles, entre les terminaux et les réseaux (signalisation dite *NAS*), et entre équipements réseaux (protocoles SIGTRAN).

Cet ensemble de fonctionnalités est mis en œuvre dans `pycrate_corenet` qui réalise les fonctions principales d'un cœur de réseau mobile 3G et 4G, et permet ainsi de prendre en charge les connexions d'antennes-relais et de terminaux mobiles, tout en gardant un contrôle fin du comportement des multiples piles protocolaires.

3 Focus sur l'ASN.1

Très peu de compilateurs ASN.1 en source ouverte (aucun ?) proposent l'ensemble des fonctionnalités supportées par Pycrate. Les compilateurs ASN.1 en source ouverte les plus complets sont `asn1c` de Lev Walkin, compilateurs générant du code C, qui ne supportent malheureusement pas officiellement le codec PER aligné et ne propose qu'un support incomplet (quoiqu'en cours d'extension ces derniers mois) des objets de type CLASS, et le compilateur fourni par le langage Erlang, qui ne supporte apparemment pas le codec PER non aligné.

La compilation d'une spécification ASN.1 présente de nombreux avantages. Elle permet de prédéfinir (dans une certaine mesure) les types et tailles d'objets dont les valeurs vont ensuite être sérialisées via les encodeurs. Elle permet également d'identifier au sein du modèle de données les

dépendances entre différents objets ainsi que des constructions dangereuses. Les sections qui suivent proposent quelques exemples :

3.1 Les objets non bornés ou étendus

Par défaut en ASN.1, ni les entiers, ni les chaînes n'ont de bornes ; en conséquence, lorsqu'une spécification n'explicite pas les bornes de tels objets, ceux-ci doivent pouvoir contenir (en théorie) des valeurs infiniment élevées ou longues !

```
MyInt1   ::= INTEGER
             -- pas de borne sur la valeur
             -- ni la taille de l'objet
MyInt2   ::= INTEGER (0..4096)
             -- bornes sur les valeurs,
             -- entrainant une borne sur la taille
MyObjStr ::= OCTET STRING (SIZE (4..32))
             -- bornes sur la taille de la chaîne d'octets
```

De plus, les objets qui listent un contenu séquentiel, telles les énumérations ou les séquences, lorsqu'ils sont extensibles, peuvent alors transporter des valeurs indéfinies (et pour le coup indéfiniment longues...).

```
MyEnum ::= ENUMERATED {orange, rouge, bleu, ..., vert)
           -- énumération étendue, pouvant embarquer
           -- une quelconque valeur indéfinie
MySeq  ::= SEQUENCE {
  myInt1 MyInt1,
  myInt2 MyInt2,
  myEnum MyEnum,
  ...,
  myStr  MyObjStr
  }        -- sequence étendue, pouvant embarquer
           -- des valeurs supplémentaires indéfinies
```

Pycrate bénéficie tout d'abord du moteur Python concernant la gestion des entiers longs et de traitements optimisés pour les chaînes. De plus, le moteur d'encodage ASN.1 supporte la fragmentation dans les encodeurs PER, BER et CER, ainsi que les extensions non définies.

Ainsi, il permet d'une part d'identifier facilement de tels objets dans les spécifications protocolaires via le compilateur, et d'autre part d'encoder et de décoder de nombreux cas rares ou limites des protocoles ASN.1.

3.2 Les structures récursives et complexes

Certains protocoles utilisent des constructions récursives ou circulaires. Ce type de construction est pris en charge par Pycrate, et il est ainsi possible d'affecter des valeurs avec une profondeur très élevée et d'encoder le résultat, sous réserve d'une adaptation adéquate du niveau de récursion autorisé dans l'interpréteur Python.

```
MyRecSeq == SEQUENCE {
  mySeq    MySeq,
  myRecSeq MyRecSeq OPTIONAL
  }          -- exemple de sequence récursive,
             -- son auto-appel doit obligatoirement rester
             -- optionel
```

Plus largement, certains formats sont tellement complexes qu'ils peuvent atteindre un niveau de profondeur assez important, sans faire appel à la récursion. La manipulation de la version compilée de tels formats permet de mettre au jour facilement ce type de complexité en utilisant un algorithme de recherche simple parcourant les références entre objets.

Par exemple, la spécification ASN.1 du protocole de signalisation entre un modem et un contrôleur radio UMTS définit un message `ActiveSetUpdate` utilisé dans certaines procédures de *hand-over* (basculement d'une antenne-relais à une autre lorsqu'une communication est en cours). Ce message référence 6247 types basiques (nombres, chaînes, énumérations) et a une profondeur maximale de 34 niveaux. Un test avec la bibliothèque permet de visualiser cette complexité :

```
>>> from pycrate_asn1dir import RRC3G
>>> RRC3G.PDU_definitions.ActiveSetUpdate.get_complexity()
(6247, 34)
>>> RRC3G.PDU_definitions.ActiveSetUpdate.get_proto()
[...]
```

3.3 Les types *ouverts*

Des objets spéciaux permettent de spécifier des types *ouverts*, qui ne sont alors déterminés que lors de l'exécution (et non de la compilation), via l'usage de tables de correspondances. Les types de chaîne de bit ou d'octets peuvent aussi *contenir* un autre type ASN.1 arbitraire.

Les normes ASN.1 fourmillent de cas spéciaux, voire de pièges, qui sont beaucoup plus faciles à éviter lorsque le développeur s'appuie sur une version compilée de la norme, et non sur sa simple compréhension humaine de la syntaxe décrivant les formats.

3.4 Les encodeurs standards

L'encodage PER *(Packed Encoding Rules)*, aligné ou non sur l'octet, peut paraître complexe : il s'appuie sur les bornes des objets pour produire une sérialisation compacte. Ainsi, la longueur en nombre de bits de valeurs à encoder sera limitée au nombre de bits maximum nécessaire pour encoder toutes les valeurs possibles entre la borne basse et la borne haute : dans ce cas, la longueur d'une valeur sérialisée d'un objet borné est fixe. De plus, PER n'encode pas le type des objets (via des *tags*), c'est pourquoi il est nécessaire de compiler une spécification ASN.1 si on veut l'utiliser avec un encodeur PER. En effet, les types des objets n'étant pas apparents dans l'encodage, le codec doit obligatoirement fonctionner en disposant de la connaissance du modèle de données a priori.

L'encodage BER *(Basic Encoding Rules)* peut paraître plus simple au premier abord, mais il recèle de subtilités qui peuvent entraîner des problèmes dans les différentes implémentations. Le principe de base de BER est d'encoder chaque valeur dans une structure de type *Tag-Length-Value*, cependant la norme BER définit de nombreuses options et possibilités distinctes pour l'encodage d'une même valeur, ou même du préfixe de longueur *Length*. De ce fait, deux encodeurs sont dérivés de BER afin de proposer des encodages canoniques : l'encodeur CER *(Canonical Encoding Rules)* et l'encodeur DER *(Distinguished Encoding Rules)*. En BER, CER ou DER, le fait que toutes les valeurs soient systématiquement *taggées* (quoique parfois *implicitement*) et préfixées par leur longueur, permet de pouvoir effectuer un décodage sans connaître la spécification a priori. Ceci est à double tranchant, comme on peut l'observer avec les très nombreuses applications utilisant le format X.509 sans le manipuler dans sa version compilée, mais en travaillant directement avec des encodeurs « en dur » et des décodeurs « aveugles »...

Pour toutes ces raisons, et bien plus encore, l'usage d'un compilateur ASN.1 est salvateur pour le développement d'applications solides, mais aussi pour le test et l'analyse de systèmes exposant des interfaces spécifiées en ASN.1.

4 Autres modules de Pycrate

Au-delà de ces fonctionnalités télécoms, quelques autres modules peuvent être utiles : `pycrate_ether` fournit les structures permettant d'encoder et de décoder les protocoles Ethernet et IPv4/IPv6 de base, et `pycrate_media` supporte quelques formats multimédia courants, tels que JPEG, GIF, TIFF ou MPEG. Ce dernier est utilisé dans l'application

`pycrate_showmedia` qui affiche la structure détaillée d'un fichier multimédia.

5 Pour aller plus loin

Outre l'exemple du cœur de réseau mobile 3G et 4G, on peut facilement réaliser des applications simples, qui s'interfacent rapidement sur des systèmes télécoms. On peut imaginer :

- requêter un HLR en définissant une application simple mettant en œuvre une pile M3UA/SCCP et TCAP/MAP ;
- décoder et analyser les trames radio *loggés* par les modems cellulaires à l'aide de leur interface de diagnostic ;
- simuler la connexion d'antennes-relais et de terminaux mobiles sur des équipements de cœur de réseau mobile à tester...

6 Disponibilité de l'outil

Pycrate est d'ores et déjà en source ouverte, publié sous licence GPL[1]. Le fichier `README.md` donne toutes les explications pour l'installation, qui n'est même pas strictement nécessaire tant que le contenu du répertoire principal est dans le chemin de l'interpréteur Python. Un wiki[2] présente certaines des fonctionnalités les plus intéressantes avec des exemples concrets d'utilisation.

La bibliothèque et son wiki évoluent régulièrement, depuis leur publication initiale en juillet 2017.

7 Conclusion

De très nombreux systèmes télécoms mettent en œuvre des interfaces complexes, s'appuyant sur des protocoles difficiles à appréhender. D'un autre côté, très peu d'outils en source ouverte sont disponibles pour pouvoir évaluer la sécurité de telles interfaces. La plupart de ces outils sont développés en C (osmocom, openbts, srs-lte...), voire en Erlang... et ne sont pas pensés pour permettre des tests à la limite des spécifications protocolaires, voire en violation de celles-ci.

Dans ce contexte, Pycrate est un outil unique, proposant des fonctionnalités avancées tout en restant simples à mettre en œuvre, et utiles à quiconque devant travailler avec des équipements cellulaires et télécoms.

[1] `https://github.com/anssi-fr/pycrate`
[2] `https://github.com/ANSSI-FR/pycrate/wiki/The-pycrate-wiki`