Se connecter

Proposer un contenu

Identifiant
Identifiant

Mot de passe
Mot de passe

☐ Connexion automatique

Se connecter

Pas de compte ? S'inscrire...

10 jan. 2026

8

« It works on my satellite » ou l'histoire d'un bug dans l'espace

Posté par **2PetitsVerres** (**Mastodon**) le 10 janvier 2026 à 10:59.
Édité par **6 personnes**. Modéré par **Benoît Sibaud**. **Licence CC By-SA.**
Étiquettes : **satellite**, **bug**

Cette dépêche raconte un vieux bug que j'ai eu sur un satellite. L'identification, la reproduction, la correction. C'est le bug qui m'a le plus intéressé/marqué dans ma carrière (jusqu'ici), C'est pourquoi cela pourrait aussi vous intéresser.

L'appel

Il y a bien longtemps, dans une galaxie lointaine. Ah non, pardon. Un long weekend de 14 juillet, sur une plage, je reçois un coup de fil : « *Un des satellites a rebooté, à cause d'une erreur logicielle, est-ce que tu es disponible pour venir comprendre ce qu'il s'est passé ? A priori, il fonctionne toujours, mais il est passé tout seul sur le calculateur redondant.* »

Quelques mois avant, on avait lancé une première grappe de six satellites ; d'autres lancements sont prévus pour compléter une constellation dans les mois/années à venir. Comme tout marche bien depuis des mois, personne de l'équipe logiciel de bord n'est d'astreinte. Sur ces satellites, j'étais surtout sur la partie validation. En gros, ce jour-là pour moi, ce n'était pas possible, mais j'y suis allé le lendemain, un samedi ou dimanche.

Sommaire

L'objectif et les moyens de débug

Si nos managers nous ont appelé, c'est parce quand un satellite bugue en prod (on va dire en vol, plutôt), c'est comme pour n'importe quel autre logiciel, *des gens* veulent des réponses à des questions comme :

- pourquoi ?
- est-ce que c'est grave ?
- est-ce que ça va se reproduire ?
- comment on corrige ?

Par contre, les moyens sont potentiellement différents de ce que vous avez dans d'autres environnements (ou pas, j'imagine que ça dépend des gens) Ce qu'on a :

- le code
- la doc
- des bancs de tests (avec le même matériel pour le calculateur)
- des gens
- un tout petit peu de contexte logiciel sauvegardé au moment de l'erreur (j'y reviens)
- la télémétrie avant l'anomalie (tout allait bien)
- la télémétrie après l'anomalie (tout va bien, mais on est passé du mode matériel 2 au mode 3. En gros c'est le même, sauf qu'on utilise certains équipements "redondants" au lieu du "nominal", dont le calculateur)

Premier élément, qui a mené au fait que c'est nous (du logiciel) qui avons été appelés, c'est que le matériel qui gère le mode (2 -> 3) peut changer de mode pour plusieurs raisons, mais il sait pourquoi il le fait. Et la raison c'est « le logiciel m'a dit de le faire ». Donc ça vient de nous.

L'analyse

Comme tout va bien, on va regarder le contexte sauvegardé. Ce n'est pas un *core dump* qu'on peut passer à `gdb`, mais ça contient quelques infos :

- le code de l'erreur `ILLEGAL CPU INSTRUCTION`
- le `Program Counter` `%pc` qui nous donne l'adresse de l'instruction exécutée au moment de l'erreur
- l'adresse de la prochaine instruction à exécuter `%npc` (ici c'est l'adresse juste après `%pc`, rien de surprenant)
- une copie des registres (bon, on ne va pas en avoir besoin, donc je ne vous fais pas un cours sur SPARC et ses registres tournant, de toute façon j'ai oublié. On pourrait probablement les utiliser pour récupérer partiellement la pile d'appel, on l'a surement fait)
- la date et l'heure (super info utile. Enfin, ça correspond à notre anomalie, j'imagine que c'est pour ça qu'on l'avait)
- surement d'autres choses, mais pas utiles pour la suite.

Problème résolu donc ? on est à l'adresse `%pc`, on l'exécute et le CPU nous dit que l'instruction n'est pas légale. Qu'est-ce qu'il y a ici ? Une instruction légale, quelle que soit la valeur des registres. Pareil pour un peu plus haut et un peu plus bas, rien qui provoque cette erreur. Que s'est-il passé ?

On est dans l'espace, donc l'explication facile (dès qu'on n'explique pas un truc) : l'instruction a dû avoir un *Single Event Upset* (SEU), un bit flip. Ça a transformé une instruction légale en instruction illégale. C'est simple ? Sauf que non, on est dans l'espace, en conséquence, on a tout un mécanisme de protection contre les SEU. C'est pas infaillible (par exemple si on a deux bits inversés, on ne peut pas corriger) mais ce n'est pas la bonne signature. Si c'était ça, ça dirait `DOUBLE EDAC ERROR`, pas `ILLEGAL CPU INSTRUCTION`.

Donc la cause de l'anomalie n'est pas un SEU.

EDAC / Protection contre les SEU

Je suis sûr que vous êtes intéressé, donc je vais vous décrire la protection contre les bit flips. C'est un mix de matériel/logiciel (en plus d'avoir une boite autour qui diminue la probabilité). En mémoire (RAM, ROM) pour 4 octets de données "utiles", on consomme 5 octets. Le 5ᵉ octet contient un code de contrôle calculé à partir des 4 autres (EDAC ᵂ). Si un bit change (sur les 5 × 8 = 40 bits), on peut non seulement le détecter mais aussi reconstruire la valeur correcte. Si deux bits changent (ou plus, mais il y a une limite), on peut détecter l'erreur mais pas la corriger (cf: le `DOUBLE EDAC ERROR` mentionné plus haut)

C'est complètement transparent vu du logiciel (code source, ou assembleur), tout ça est calculé par le matériel. Quand on écrit en mémoire `0x12345678` il calcule le code et écrit `0x12345678XY` avec la bonne valeur de X et Y. Quand on lit, pareil, le matériel commence par lire `0x12345678XY`, calcule la somme de contrôle sur les 4 octets, si c'est le bon, il nous donne `0x12345678`.

Là où ça se complique, c'est quand il y a un changement. Disons qu'on a maintenant `0x02345678XY`. (1 --> 0). Il se passe deux choses ici :

1. le matériel dit au logiciel `0x12345678` (il corrige, mais uniquement la valeur envoyée au software. Pas la valeur enregistrée en mémoire)
2. il émet un signal `SINGLE EDAC ERROR`.

C'est là que le logiciel intervient, dans le point 2. Ce signal est lié à une *trap* qui corrige la mémoire. Schématiquement c'est lié à une fonction qui ressemble à ceci (en assembleur SPARC en vrai, mais j'ai tout oublié)

```
; adresse vient du contexte, c'est l'adresse c
disable_edac_trap: ; Désactiver la trap. Sinon
load [adresse], reg ; Lire 4 octets (lecture =
enable_edac_trap: ;
store reg, [adresse] ; Réécrire la valeur corr
```

On lit la valeur, c'est corrigé vu du logiciel par le matériel, on réécrit la valeur, tout est corrigé.

Cette trappe peut être déclenchée par n'importe quelle instruction qui lit de la mémoire (ou par le fait de charger une instruction elle-même depuis la mémoire), et on a même une tâche de fond (plus basse priorité, qui tourne en permanence quand il reste du temps de calcul disponible) qui fait

```
// en gros. En vrai légèrement plus compliqué
void background_task(void) {
int address = MEMORY_START;
volatile int value;
while (1) {
value = *address; // s'il y a un bit flip en m
address += 4;
if (address >= MEMORY_END) {
address = MEMORY_START;
}
}
}
```

L'idée de cette fonction c'est de lire la mémoire régulièrement. Si on ne faisait pas ça, peut-être que certaines cases mémoires auraient deux bit flips, car pas corrigé après le premier si on ne lit pas la mémoire avant qu'un autre arrive. Ce n'est pas très fréquent d'avoir des bit flips, mais sur les 6 satellites, en cumulé, on en détecte quelques-uns par jour.

L'hypothèse

De retour à la case départ donc. On exécute apparemment l'instruction stockée dans `%pc`, valide. Et le CPU nous dit qu'elle est invalide, mais clairement, elle est valide. On tourne en rond, on est samedi ou dimanche, fin d'après midi, et le satellite, lui aussi il tourne en rond, sans problèmes. Tout à coup, quelqu'un a l'idée de dire « bon, on ne résoudra pas ça aujourd'hui. On se revoit lundi ? ». On rentre, je bois un verre avec mes colocs (enfin, je suppose. C'était une activité habituelle pour un weekend, ça, au moins)

Retour au bureau, et là (surement plus tard, pas lundi 9h) on a David (un collègue) qui propose : "*Comme clairement `%pc` est valide, est qu'on exécute quelque chose d'invalide, est-ce qu'on est sûr qu'on a bien enregistré `%pc`?*". On vérifie, le code qui fait ça a l'air correct. En plus le contexte général, ce qu'il y a dans les registres est correct. Toujours David "*OK, le logiciel est correct, mais est-ce qu'on est sûr que `%pc` c'est bien toujours l'instruction qu'on exécute ?*".

Donc, on vérifie, par acquit de conscience et on remarque que non, pas nécessairement. Si on est dans une trap, le `%pc` qu'on enregistre pointe vers l'instruction qui a provoqué la trap, pas l'instruction de la trap qu'on exécute. Bon, OK, ça ne nous avance pas nécessairement (mais si j'en parle…)

Nouvelle question donc : Si on est à `%pc`, quelles sont les traps qui peuvent s'exécuter ? Il y a plein de possibilités, la plupart viennent de causes extérieures (timer matériel, plein d'autres évènements extérieurs) et potentiellement aussi la trap de l'EDAC si on lit une valeur (et l'instruction à `%pc` lit une valeur).

Donc techniquement, on pourrait aussi être n'importe où dans le code (assembleur) de toutes les traps. Avant on cherchait pourquoi c'était illégal d'exécuter `%pc`, maintenant on cherche pourquoi ça serait illégal d'exécuter `%pc` ou n'importe quelle ligne d'une trap active/activable à ce moment-là.

Chez moi, ça marche

Sauf que le code des traps, c'est pas nous qui l'avons écrit. C'est bien du code qui vient de l'entreprise, mais il existe depuis plusieurs années, est utilisé sur le même processeur depuis plusieurs années, et il a plusieurs dizaines d'années de vol (cumulé, en additionnant les satellites) sans problème.

En suivant les principes bien connus du développement logiciel, si on utilise un logiciel sur étagère, pas besoin de le valider (surtout ça coute de l'argent. Cela dit même si on avait essayé, je ne pense pas qu'on aurait trouvé de problème), vu qu'il marche. Par acquit de conscience, on demande, et on nous répond "*bah chez nous ça marche*" (la légende veut qu'une histoire similaire soit à l'origine de Docker, je ne sais pas si c'est vrai, mais le fameux "*it works on my desktop, ship my desktop*"…)

Vous avez peut-être lu le titre de l'article, donc vous imaginez où je vais. On se demande « OK, pourquoi ça marche pour eux, et pas pour nous ? » Quelles sont les différences ?

- on est sur le même CPU/MCU (donc non, c'est pas ça)
- on a changé de compilateur pour une nouvelle version (mais 1. c'est un compilateur "certifié", et 2. les traps sont en assembleur…)
- on est en orbite plus basse, et on a plus de SEU (mais même, quand on regarde leur historique, ils en ont beaucoup aussi, et en cumulé, beaucoup plus. Après… peut-être n'a-t-on pas de chance ?)

L'erreur

Ok, on a changé de compilateur, les traps sont en assembleur, mais le reste du code est dans un langage bien plus courant (non, je rigole, en vrai c'est en Ada…), peut-être que l'interaction entre les traps et le reste du code a changé ?

Pourquoi est-ce qu'on a décidé de changer de compilateur ? Ah pour des histoires de taille mémoire (*640 kB should be enough*? On avait même plus, genre 2 Mo de ROM, 4 Mo de RAM, large… ou pas). D'ailleurs, au moment du changement, on en a profité pour faire quelques optimisations. Non pas des flags genre `-01` ou `-02`. Plus des choses sur le layout mémoire, on a ajouté `__attribute__((packed))` qui est supporté, on a un peu changé le linker script…

Par exemple, le `packed`, ça nous permet de gagner de la place, avant toutes les variables étaient alignées sur une adresse multiple de 4, que ça soit un nombre sur quatre octets, ou un `char` d'un octet, ils prenaient au moins quatre octets. Maintenant, on a mis les data types multiples de quatre au début de la structure, bien alignés, puis les types qui prenent deux octets, on en met deux dans quatre octets (au lieu d'un et de gacher deux octets pour rien), puis les types de un octect, on en met 4.

D'ailleurs, par exemple, l'instruction à `%pc`, elle charge une donnée d'un seul octet qui est dans une adresse du type `XXX+3`, où X est un multiple de 4. C'est pas illégal de faire ça (donc non, toujours pas d'instruction illégale ici)

Après quoi, c'est là où David revient (dans mon souvenir en tout cas, ça venait beaucoup de lui, mais on était beaucoup à échanger sur le sujet). "*Ok, `%pc` lit une donnée non alignée, et il le fait correctement. Mais s'il y a un bit flip, il se passe quoi ?*. Bah rien, EDAC détectée, trap, on exécute le code assembleur qui marche sur les autres satellites.

Ah oui, mais non. Si on lit un octet, on peut lire `XXX+3`, mais si on lit 4 octets, c'est interdit. Il faut lire une adresse multiple de 4. Et donc on a une EDAC, et quand on rentre dans la trap

```
; adresse == XXX+3
disable_edac_trap: ;
load [adresse], reg ; Lire 4 octets
enable_edac_trap: ;
store reg, [adresse] ;
```

Ah oui, mais non. `load` ça lit 4 octets, c'est illégal de lui passer une adresse non multiple de 4, c'est une illegal instruction. Donc ça pourrait être ça :

1. bit flip sur les quatre octets situés à `XXX` (l'EDAC est toujours calculé sur 4 octets d'une adresse alignée, même si on lit décalé)
2. on rentre dans la fonction qui contient `%pc`
3. on lit un octet à `XXX+3`
4. ça déclenche la trap
5. la trap essaye de lire 4 octets à `XXX+3`
6. `ILLEGAL CPU INSTRUCTION`, allez en prison sans passer par la case départ

La reproduction

Sur le papier, ça marche. On peut même faire un petit logiciel sur le banc, qui fait juste un `load [XXX+3], reg` et qui génère une `ILLEGAL CPU INSTRUCTION`. Mais évidemment nos managers (et notre client) voudraient un peu plus qu'un « sur le papier, c'est ça, trust me bro ».

Donc la question "*c'est possible de reproduire exactement comme dans l'espace, plutôt que de juste exécuter une instruction illégale à la main ?*". Avec le vrai logiciel qui était dans l'espace, pas un logiciel de test ?

Bien sûr, il suffit d'attendre d'avoir un bit flip, sur le banc, juste au bon endroit, au bon moment. Vous avez combien de siècles devant vous ? Ou alors est-ce qu'on peut mettre le banc à côté d'un réacteur nucléaire ? Ça devrait accélérer les choses (du bon côté du mur de confinement. Ici, “bon”, ça veut dire mauvais pour les humains)

On va quand même regarder si on peut provoquer un bit flip autrement. Bon, a priori, en interne, au logiciel, on ne sait pas comment faire. La doc du processeur (qui vient avec l’edac) ne nous aide pas non plus. On demande à ceux qui nous ont dit que « chez eux, ça marche » qui nous répondent que la trap de l’edac, ils ne l’ont jamais testé, c’est juste une revue de code.

Bon, on envoie quand même un courriel au fabricant du proc, au cas où. Réponse rapide « je reviens vers vous dès que je sais ». Quelques jours (2, 3 semaines ?) plus tard : "*Ah oui, c’est possible. D’ailleurs c’est documenté. Page WSYZ sur 5000, il y a **un* paragraphe qui explique comment faire*".

Le TL/DR du paragraphe : Il est possible de désactiver l’EDAC en écriture. Par contre il faut faire des choses spécifiques, donc on a pas de commande prévue pour le faire “simplement” depuis l’extérieur, il faudrait une nouvelle fonction.

```
void generer_bit_flip(int address, int valeur)
*address = valeur; // écrit la valeur correcte
manipulate_specific_register_to_disable_edac()
*address = valeur ^ 0x00000001; // écrit la va
manipulate_specific_register_to_enable_edac();
}
```

Ça tombe bien, le logiciel qui est dans l’espace a deux fonctionnalités qu’on a testé, mais jamais en vrai avec un truc vraiment utile

1. on peut patcher la mémoire et écrire ce qu’on veut, où on veut (code, données)
2. on a plusieurs “fonctions” périodiques qui ne font rien, et qui sont prévues pour être patchées si on veut ajouter quelque chose (via la fonction de patch plus haut)

Donc on peut créer une fonction comme ça (en gros)

```
void generer_bit_flip(int address, int valeur)
static int actif = TRUE;



if (actif) {
*address = valeur; // écrit la valeur correcte
manipulate_specific_register_to_disable_edac()
*address = valeur ^ 0x00000001; // écrit la va
manipulate_specific_register_to_enable_edac();
actif = FALSE; // on ne veut le faire qu’une f
}
}
```

Une fois qu’on a la fonction, on la compile. Ensuite on charge le logiciel normal sur le banc, on se met en conditions « avant l’anomalie », on uploade la fonction, on l’active et…

Le banc change de mode, passe du mode 2, au mode 3, sur le calculateur redondant. On vérifie le contexte, même signature que l’anomalie en vol. C’est bon on a fini. (Ouf, mon journal est déjà trop long)

La correction (Over-The-Air, mais sans l’air)

Oui, non, pas exactement. On a une explication, il faut une correction maintenant. Bon, c’est simple. Pour lire une adresse alignée sur 4, il suffit de mettre deux bits à 0. Finalement, voilà le patch

```
address = address & ~0x3 ; ** Cette ligne est
disable_edac_trap: ;
load [adresse], reg ;
enable_edac_trap: ;
store reg, [adresse] ;
```

Oui, c’est un patch d’une instruction dans le binaire. (Techniquement, 5 instructions, parce qu’il faut décaler les 4 instructions existantes de 1, mais on avait des noop en dessous, donc ça rentre)

La dernière question, c’est quelle stratégie d’ *update* appliquer. On a techniquement quatre familles de satellites à considérer :

1. les satellites « pré-existants », qui utilisent l’ancien compilateur, sans packed et déjà dans l’espace.
2. le satellite qui a eu l’anomalie.
3. les 5 autres satellites de la grappe.
4. les futurs satellites, non lancés.

Ce qui a été décidé : La première catégorie : Techniquement, on pourrait discuter du fait qu’il y a un bug ou non. Mais même si on considère qu’il y a un bug, il ne peut pas être déclenché. Donc on ne touche à rien. La catégorie 4, c’est facile. Ils sont au sol, on fait une nouvelle version complète du logiciel, on reflashe la rom en entier, et on vérifie.

Il reste les deux autres catégories. Bon la seule différence, c’est qu’un, toujours en mode 3, tourne pour l’instant sur le calculateur redondant (on peut revenir en mode 2, manuellement, si on veut). Donc on décide « on va faire la même chose », et on va corriger le problème (on aurait pu ne rien faire et dire « bah, si ça arrive, on connaît et on revient à chaque fois manuellement en mode 2 »)

Là encore, même si on corrige, on a plusieurs choix :

1. Mettre à jour la ROM. En fait non, les ROM, parce que chaque calculateur a la sienne. Et le nominal ne peut pas écrire la ROM du redondant, et inversement. (Dès lors, si on veut patcher, qu’est-ce qu’on patche ? Le deux ROM ? Faut-il reconfigurer à la main pour rebooter sur le redondant ?)
2. utiliser un mécanisme prévu pour « patcher, mais sans patcher la ROM ».

La solution 2, retenue, c’est un mécanisme (déjà dans le logiciel) qui permet de mettre les infos dans une autre mémoire (partagée par les deux calculateurs). Au boot, la ROM est copiée dans la RAM (on exécute le code depuis la RAM), et « avant de démarrer » on vient regarder dans cette table, si l’on doit patcher la RAM. Cela donne quelque chose comme :

ROM (logiciel original) --> Copie vers la RAM --> RAM (logiciel original) --> fonction de patch au boot, vient modifier la RAM --> RAM (trap corrigée) --> boot du logiciel.

Conclusion

Qu’est-ce que je retiens principalement ?

- quand on me dit que du code fonctionne, donc qu’il est correct… j’ai un doute
- Ce n’est pas parce que la doc explique quelque chose qu’on peut le trouver. Surtout quand elle fait 5000 pages… Il ne faut pas hésiter à demander

Voila, en quelques pages, une vieille histoire qui m’a marqué. Je suis probablement une des personnes qui a participé à un des patchs le plus haut du monde (plus de 1 000 km d’altitude)

Bon en vrai, la NASA fait des mises à jour logicielles sur des rovers sur Mars, donc c’est clairement pas le record mais c’est pas trop mal (ils ont même peut-être des mises à jour sur leurs sondes plus loin de la terre)

Note : cette histoire date maintenant d’il y a plus de dix ans. Il y a donc forcément des simplifications, des imprécisions, et probablement des erreurs. Aucun satellite n’a été maltraité pendant cette enquête. Il y en a bien un qui est tombé à terre, mais ça c’était avant le lancement.

Aller plus loin

- Journal à l’origine de la dépêche (14 clics)

(1 commentaire). Markdown EPUB

Intéressant

Posté par **Bruno** (**Mastodon**) le 10 janvier 2026 à 11:17.
Évalué à 2 (+1/-0).

> Merci du partage.
>
> Je chipote mais les exemples d’assembleur sont du macro-assembleur.
> On me murmure dans l'oreille qu'il n'y a plus que ça depuis longtemps.
> Bon sang le monde change, j'ai commencé en programmant en hexadécimal…

Répondre

Envoyer un commentaire

Suivre le flux des commentaires

Note : les commentaires appartiennent à celles et ceux qui les ont postés. Nous n’en sommes pas responsables.

Revenir en haut de page

Derniers commentaires

- Vielle alarme / optio…
- James Webb
- Re: Je crois que j'ai c…
- Slicer
- Re: Je crois que j'ai c…
- NON
- Re: ia routeur
- c'est pas beau
- Au risque de me rép…
- Re: Souvenirs, souv…
- Intéressant
- Re: Je crois que j'ai c…

Étiquettes (tags) populaires

- intelligence_artificielle
- merdification
- hppa
- grands_modèles_de…
- états-unis
- sortie_version
- donald_trump
- administration_fran…
- linux
- capitalisme_de_surv…
- capitalisme
- note_de_lecture

Sites amis

- April
- Agenda du Libre
- Framasoft
- Éditions D-BookeR
- Éditions Eyrolles
- Éditions Diamond
- Éditions ENI
- La Quadrature du Net
- Lea-Linux
- En Vente Libre
- Grafik Plus
- Open Source Initiative

À propos de LinuxFr.org

- Mentions légales
- Faire un don
- L’équipe de LinuxFr.…
- Informations sur le s…
- Aide / Foire aux que…
- Suivi des suggestion…
- Règles de modération
- Statistiques
- API pour le dévelop…
- Code source du site
- Plan du site