Communication cachée via le champ magnétique émis par un ordinateur

Ludovic APVRILLE
Professeur, Télécom Paris. ludovic.apvrille@telecom-paris.fr.

Tout courant électrique émet un champ magnétique … dont celui qui alimente les processeurs. Cet article montre comment on peut, assez facilement, exploiter le champ magnétique émis par l'alimentation d'un processeur pour émettre des données. Ce canal caché a notamment été utilisé lors du CTF ph0wn pour un des challenges…

1. Une courte introduction aux champs magnétiques

Un champ magnétique est un espace autour d'un matériau magnétique ou d'une charge électrique en mouvement dans lequel une force magnétique agit. On utilise en général un ensemble de vecteurs pour représenter un champ magnétique : leur direction indique celle de la force magnétique, et leur longueur indique la force du champ présent à l'origine du vecteur.

Un peu de physique et d'histoire avant d'arriver à des aspects plus informatiques, reliés à ces champs magnétiques. La loi d'Ampère (ou Théorème d'Ampère) est un principe fondamental de l'électromagnétisme qui relie le champ magnétique circulant dans une boucle fermée au courant électrique passant par cette boucle. Formulée par André-Marie Ampère dans les années 1820, elle fournit un moyen de calculer le champ magnétique produit par un courant électrique.

L'intégrale de ligne du champ magnétique (B) autour de toute boucle fermée est égale à la perméabilité du vide (μ0) multipliée par l'intensité du courant électrique total (I) traversant la boucle. Mathématiquement, elle exprime ainsi :

$$\int_{boucle} \vec{B} \cdot d\vec{l} = \mu_0 I_{enc}$$

Cette équation indique que le champ magnétique circulant autour d'un fil conducteur parcouru par un courant peut être déterminé par la quantité de courant dans le fil et la forme du chemin encerclant le fil et la forme du chemin encerclant le fil. Cette loi peut être appliquée dans le cas d'une bobine (dont une des applications très courante est le moteur électrique), mais aussi à un fil électrique. Par exemple, le champ magnétique (B) à une distance (r) d'un conducteur droit long portant un courant (I) est donné par :

$$B = \frac{\mu_0 I}{2\pi r}$$

Cela montre que le champ magnétique autour d'un fil droit long diminue inversement avec la distance par rapport au fil (voir Fig. 1).

En informatique, les champs électromagnétiques ont plein d'applications, dont les bonnes vieilles bandes magnétiques, mais aussi les disquettes, les disques durs à plateaux, les première RAM, etc.

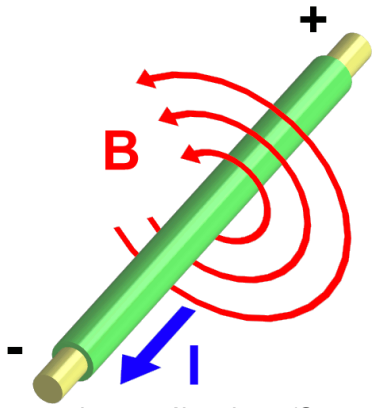

Fig. 1 : Champ magnétique autour d'un conducteur électrique (Source : Wikipédia).

2. Champs magnétiques émis par un ordinateur

Un processeur étant un ensemble de pistes traversées par des courants électriques, il émet donc, selon la loi d'Ampère, des champs magnétiques dont la force dépend de l'intensité, et de la consommation électrique. Cette consommation électrique d'un processeur dépend des calculs qu'il effectue.

Un processeur multi-coeur consommera d'autant plus d'énergies que son nombre de coeur qui effectue des calculs est élevé. Ainsi, le champ magnétique émit par l'alimentation d'un processeur dépend des calculs que ce dernier effectue. Bien entendu, le processeur lui-même émet un champ magnétique, mais il est probable que sa composante résultante soit plus ou moins nulle, car ses lignes électriques se dirigent dans tous les sens. Alors qu'en ce qui concerne son alimentation elle-même, la résultante est non nulle puisque du courant circule de la source (l'alimentation du PC vers le processeur). Aussi, un processeur consomme d'autant plus d'énergie que ses lignes électriques sont utilisées. Notamment, chaque coeur de calcul consommant de l'énergie dépendant des calculs qu'il effectue, plus des coeurs sont utilisés en parallèle, plus l'intensité d'alimentation du processeur est importante, et donc plus le champ magnétique sur la ligne d'alimentation du processeur est important. Ces champs magnétiques s'appliquent évidemment aussi à l'alimentation d'autres composants, notamment la mémoire, les SSDs, les cartes graphiques, etc.

3. Mise en pratique

Cette section explique comment simplement relever le champ magnétique de votre processeur. Cela fonctionne sur la plupart des ordinateurs portables sur lesquels nous avons essayé. Pour savoir si le champ magnétique mesure correspond bien à celui du processeur, soit il faut connaître approximativement la ligne d'alimentation du processeur, soit on peut répérer cette ligne d'alimentation en variant l'intensité des calculs et en identifiant un champ magnétique se modifiant selon l'intensité de ces calculs.

3.1 Mesure du champ

La plupart des téléphones mobiles proposent une application de type « boussole » qui utilise un capteur de champ magnétique. L'utilisation de la boussole est toutefois insuffisante : plusieurs applications mobiles, dont « Physics Toolbox » [Physics], permettent d'observer des valeurs numériques brutes mesurées par les capteurs d'un téléphone mobile, et notamment les valeurs mesurées par le capteur de champ magnétique. L'application Physics permet d'afficher ce champ magnétique en 3D, ainsi selon 3 axes x/y/z. Cette application permet en outre d'enregistrer des traces de l'évolution de ces valeurs en fonction du temps, et d'afficher les courbes correspondantes.

Une fois l'application lancée, « magnétomètre » sélectionné, il ne reste plus qu'à placer le téléphone mobile au dessus du clavier (qq centimètres maximum au dessus) et de rechercher la position opportune permettant de mesurer le champ magnétique du processeur. Essayez de lancer des processus de calculs intensifs, puis des les arrêter, puis de les relancer, ainsi de suite : vous devriez voir des variations du champ magnétique si votre téléphone est bien placé.

Fig. 2 : Posez votre téléphone à quelques centimètres de l'alimentation des processeurs pour relever un champ magnétique lié aux calculs effectués par votre processeur.

3.2 Emettre un champ

A ce stade, j'imagine que vous avez presque tous essayé de mesurer le champ magnétique émis par votre ordinateur portable. Mais comment en tirer quelque chose ? Dans le cas général, pas grand-chose, cela va apparaître comme du bruit, mais difficile d'en déduire immédiatement ce que votre ordinateur fait : cela nécessite des analyse plus poussées. Certaines attaques par ces canaux cachées ont été montrées, y compris sur le champ magnétique émis par le processeur lui-même ou d'autres types d'attaques distantes [1].

Utilisons à présent le champ magnétique du processeur pour émettre volontairement des données vers un téléphone mobile. Comme ce champ magnétique dépend notamment du nombre de coeur en cours de calcul, nous pouvons écrire facilement un code C qui active n coeurs pour émettre un « 1 », n/2 (ou une autre valeur inférieure à « n ») coeurs pour un « 0 » et qui n'effectue aucun calcul pour avoir un motif entre symbole. Le code C suivant (qui est un extrait du code global, et simplifié pour la lisibilité) prend sur la ligne de commande une chaine de caractères, calcule son train binaire (ascii) puis lance des calculs fonction de ce train binaire. Le code complet est disponible ici [2]. L'idée est ainsi de lancer plus de threads de calculs pour le symbole « 1 » que pour le symbole « 0 », et aucun thread pour l'inter-symbole. Ce code a fonctionné pour des architecture avec *hyperthreading*, ce qui veut dire que soient les threads ont été répartis sur des coeurs différents, soit la consommation d'énergie est de toute façon impactée si deux threads peuvent être exécutés sur le même coeur.

```
...
void compute(int value) {
  printf("Computing %d \n", value);
  pthread_t tid[NB_OF_THREADS];
  int threads;
  if (value == 0) {
    threads = NB_OF_THREADS / 4;
  } else {
    threads = NB_OF_THREADS;
  }
  printf("Using %d threads\n", threads);
  for (int j = 0; j < threads; j++) {
```

```
    pthread_create(&tid[j], NULL, t1, NULL);
  }
  for (int j = 0; j < threads; j++) {
    pthread_join(tid[j], NULL);
  }
  waitFor(WAIT_TIME);
}

int main(int argc, char **argv) {
  if (argc < 2) {
    printf("usage: compute <text to be sent>\n");
    exit(1);
  }
  printf("Sleeping\n");
  sleep(10);
  printf("Running\n");

  while (1) { // Main loop
    int k = 0; int i;
    while (argv[1][k] != '\0') { // Handling each character one by one
      printf("Handling %c \n", argv[1][k]);
      char *str = (char *)(malloc(sizeof(char) * 9));
      for (i = 0; i < 9; i++) {
        str[i] = '\0';
      }
      int cpt = binprn(argv[1][k], str); // Binary representation
      int dec = 8 - cpt; //padding
      if (dec > 0) {
        for (i = 8 - dec; i >= 0; i--) {
          str[i + dec] = str[i];
          str[i] = '0';
        }
      }
      printf("%s\n", str);
      int l;
      for (l = 0; l < 8; l++) {
        printf("Handling: %d ", str[l]);
        if (str[l] == '0') { // Computing depending on the character
          compute(0);
        } else { compute(1);}
      }
      k++;
    }
    sleep(5);
  }
  return 0;
```

4. Challenge pour Ph0wn

Ph0wn est un CTF qui a lieu annuellement à Sophia-Antipolis. Ph0wn est un peu différent des autres CTFs, dans la mesure où les challenges se font principalement sur des objets connectés divers, comme par exemple en novembre 2023 : une communication avec un (vrai!) satellite géostationnaire, un hydrabus (https://hydrabus.com/), etc. Dans tous les cas, le but est, comme pour les autres CTF, d'identifier un flag. Dans de le cas de ce challenge, les participants sont informés que le flag commence par « ph » suivi d'un espace.

Le challenge appelé « Magnéto » est ainsi décrit aux utilisateurs : « `Magneto has contaminated my computer with his Nokia 3120... but maybe he sent a magnetic message on it? Retrieve it, and become a real X-Pico!` »

Un ordinateur portable est en accès « libre » (photo), mais interdiction d'y toucher bien sûr :-) alors que faire ? Comme je viens de vous parler de champ magnétique, il s'agit évidemment d'exploiter le champ magnétique émis de l'ordinateur portable. Il faut donc repérer une zone particulière de l'ordinateur qui émet un champ magnétique notable et qui change au cours du temps, enregistrer ce champ, puis le décoder : là est le secret.

Pour résoudre le challenge, on a enregistré une trace en CSV à cinq colonnes : temps, magnétique-x, magnétique-y, magnétique-z et champ magnétique global « magnétique-xyz ». En observant les différentes valeurs, le champ magnétique en z est celui qui évolue avec des schémas répétitifs. En traçant sa courbe (voir

Fig 3), on peut en déduire les intervalles de valeur de champ magnétique pour un « 1 », pour un « 0 » et pour un inter-symbole.

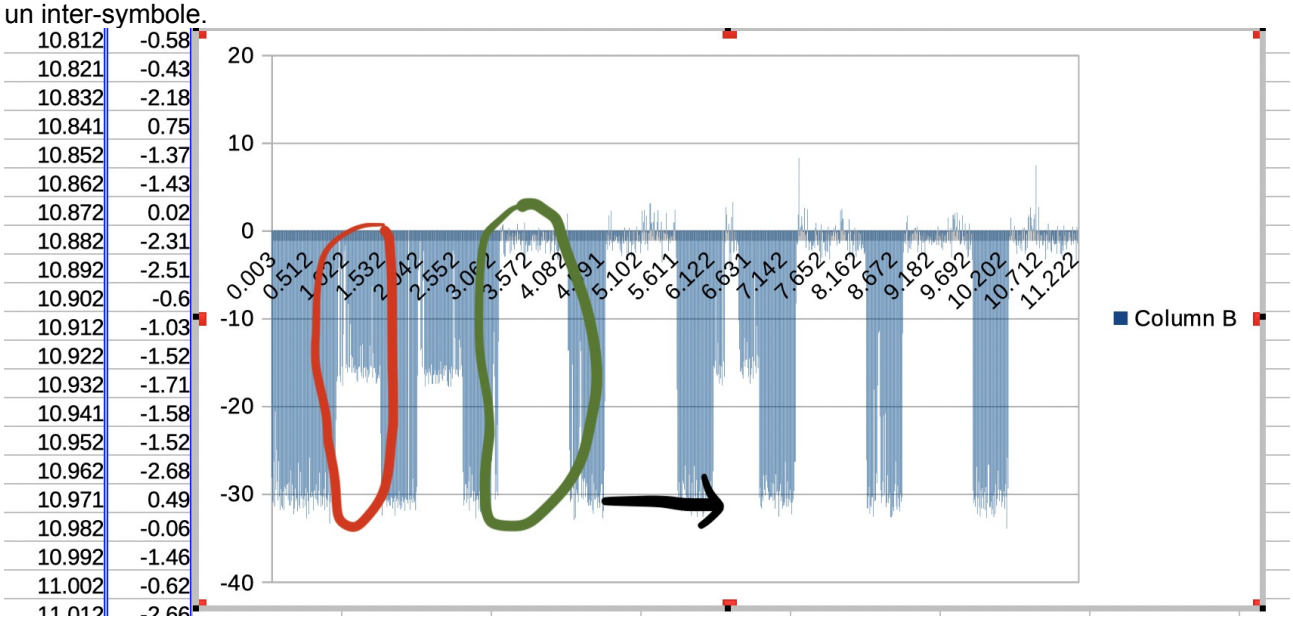

Fig. 3 : Courbe du champ magnétique vertical (z). La zone entourée en vert correspond à un « 1 », celle en rouge à un « 0 ». Les lignes verticales les plus basses correspondent à l'inter-symbole. Le code pour décoder cette trace tient compte de la forme de ces symboles : intervalles de valeurs, et longueur typique d'un symbole.

Il ne reste plus qu'à écrire un code pour reconstruire la chaine de caractère transmise. Pas de grand difficulté si ce n'est que les pics hauts ou moyens ont une longueur un peu variable, et que certaines valeurs anormales doivent être exclues. Le code Java ci-dessous permet de réaliser ce décodage du champ magnétique en CSV. Les attributs statiques de Analyzer comportent la valeur minimale du symbole 1 (ONE_MIN_VALUE), la valeur du symbole de séparation (SEPARATOR_MAX_VALUE), la durée de l'inter-symbole (TIME_BETWEEN_2_VALUES), et le nombre minimal de valeurs de « 1 » que l'on doit avoir entre deux séparateurs pour choisir si cela est vraiment un « 1 » ou pas (MIN_NUMBER_OF_ONE). Par exemple, supposons que l'on soit en inter-symbole. Avant d'avoir l'inter-symbole suivant, on va compter le nombre de valeurs supérieures à « ONE_MIN_VALUE ». si ce nombre est supérieur à MIN_NUMBER_OF_ONE alors on sélectionne un « 1 », sinon un « 0 ». Puis on recommence après l'inter-symbole suivant, etc.

```
public class Analyzer {
    public static int ONE_MIN_VALUE = -5;
    public static int SEPARATOR_MAX_VALUE = -28;
    public static int TIME_BETWEEN_2_VALUES = 1;
    public static int MIN_NUMBER_OF_ONE = 20;

    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java Analyzer file.csv");
            System.exit(0);
        }

        String pathToCsv = args[0];
        ArrayList<Data> dataList = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(pathToCsv))) {
            String line;
            boolean firstLine = true;
            while ((line = br.readLine()) != null) {
                if (firstLine) {
                    firstLine = false;
                } else {
                    String[] values = line.split(",");
                    double time = Double.parseDouble(values[0]);
                    double value = Double.parseDouble(values[3]);
                    dataList.add(new Data(time, value));
                }
            }
        } catch (IOException e) { e.printStackTrace();}
```

```
        int foundSep = 0; int foundOne = 0; double initTime = 0;
        String binary = "";
        for (Data sep : dataList) {
            System.out.println("New sep: ( " + sep.time + " , " + sep.value + " )");
            if ((foundSep == 0) && (sep.value < SEPARATOR_MAX_VALUE)) {
                foundSep = 1;
                foundOne = 0;
                initTime = sep.time;
            } else if (foundSep == 1) {
                if (sep.value > ONE_MIN_VALUE) {
                    foundOne++;
                    System.out.println("foundOne ++ ; foundOne=" + foundOne);
                }
                if ((sep.value < SEPARATOR_MAX_VALUE) && (sep.time - initTime > TIME_BETWEEN_2_VA-
LUES)) {
                    if (foundOne > MIN_NUMBER_OF_ONE) {
                        binary = binary + "1";
                    } else {
                        binary = binary + "0";
                    }
                    foundOne = 0;
                    initTime = sep.time;
                }
            }

        }
        System.out.println("binary=" + binary);
    }
}
```

En utilisant le code de décodage CSV fournit ainsi que le trace relevée, on obtenait finalement le string : « 744#99744422266663366# »

Cela ne ressemble pas vraiment à un flag ?! Ah oui, on nous parlait en énoncé d'un Nokia 3120 … Sans doute Magneto a t-il envoyé un message de type SMS en utilisant le clavier numérique de son Nokia … Donc « 7 » veut dire « p », « 44 » veut dire h, etc. On en déduit le flag de ce challenge : `ph xpicomen`.

Conclusion

Nous attendons avec impatience vos différents codes pour communiquer entre votre ordinateur et votre téléphone à plus haut débit que notre code !

Remerciements

Tous mes remerciement vont à Philippe Jaillon (Ecole des Mines de Saint-Etienne) qui m'a initié à ces canaux cachés, et à Bastien Sultan (Télécom Paris) et Axelle Apvrille (Fortinet) qui ont patiemment testé mon challenge.

Références

[Physics] Physics Toolbox, application mobile. Android : **https://play.google.com/store/apps/details?id=com.chrystianvieyra.physicstoolboxsuite**

[1] Genkin D, Pipman I, Tromer E (2015). "Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs". Journal of Cryptographic Engineering. 5 (2): 95–112. doi:10.1007/s13389-015-0100-7

[2] Code C pour transmission d'information par champ magnétique : https://perso.telecom-paristech.fr/apvrille/docs/magnetic.c