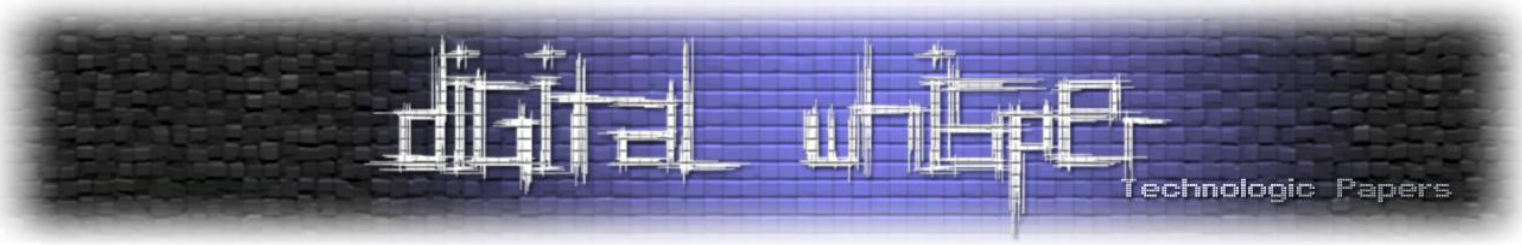


HTTP/2 - הבנה וניטור של תקשורת העתיד

נכתב ע"י ישראל (Sro) חורז'בסקי, CTO, AppSec Labs)

פרולוג

כל המשפחה התיישבה רגל על רגל והסתובבה לכיוון הקיר. המסך ירד והמקרן החל לפעול. לשבת בבית קפה שהוקצה במיוחד ליום שלם של עבודה משותפת ומהנה, להזמין קצת נשנוש ושתייה ובמשך שעה שלמה לשמוע רק הרצאות על נושאים טכנולוגיים חדשים שהיו ארוכים מכדי להיכנס למייל. תענוג.

המאמר מכיל את אחד הנושאים שהוצגו ב-Tech talk הפנימי של משפחת AppSec בחודש האחרון, והוא מכיל 2 חלקים: הסבר על מה שהשתנה ב-http/2, ואיך לראות תעבורה של http/2 (רמז: לא באמצעות Burp/fiddler).

פינת היסטוריה - SPDY is dead

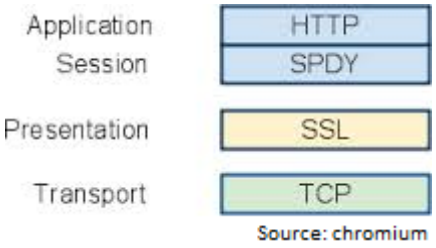
מי שלא מתלהב מהיסטוריה, יכול לעבור לעמוד הבא ולקרוא ישירות על HTTP/2. למי שכן, נעשה מעט סדר כרונולוגי.

התקן של פרוטוקול HTTP/1.1 התפרסם בשנת 1997, ומאז לא השתנה עד השנה. במובן הזה, HTTP2 (ליתר דיוק HTTP/2) הוא אחד השינויים הכי גדולים בפרוטוקול שהיו בעשרים השנים האחרונות.

בשנים שחלפו מאז שהוגדר התקן של HTTP 1.1, עולם האינטרנט השתנה משמעותית, וכיום כל דף אינטרנט שעולה בדפדפן שולח המון בקשות לשרת (ajax, css, images, ico, js, cors ועוד). כך שמעבר לעובדה שהפרוטוקול טקסטואלי - מה שיוצר בקשות גדולות (וזה בלי להגיד מילה על ה-abuse שעושים לקוקיז כשמכניסים אליהם כמויות של מידע...), כיוון שיש הרבה חיבורים מול השרת נוצר בדפדפן "תור" של בקשות. הדפדפן מגביל את כמות החיבורים פר שרת כדי לא ליצור עומס, וכשיש כמות חיבורים מוגבלת, גם כשמגדירים Connection: keep-alive, בקשה חדשה תישלח רק כשהתשובה של הקודמת מסתיימת להגיע.

בקיצור, העסק עובד לאט וצריך לעשות משהו.

זה מה שחשבו בגוגל, ולכן הקימו צוות מחקר שיצר את פרוטוקול SPDY (קוראים את זה: ספידי) שביצע מספר אופטימיזציות על שיטת התעבורה של HTTP. אחת מהן לדוגמא היא ה-Server push, שבה השרת יכול לשלוח לדפדפן קבצים לקאש, עוד לפני שהדפדפן מבקש ממנו. וכך כשהדפדפן יצטרך את הקבצים הללו, הם כבר ישכנו אצלו לבטח במטמון.



התקן של גוגל היה כל כך טוב, שהוא תפס מהר מאוד וכולם תמכו בו דפדפנים ושרתים כאחד, וכשהחליטו להגדיר את תקן HTTP/2 כולם המליצו לבסס אותו על SPDY. למה להגדיר את HTTP/2 ולא להמשיך עם SPDY? כיוון שתקן צריך להיות מוגדר ע"י גוף עצמאי ולא חברה מסחרית שהיא צד במשחק. ויאמר לשבחה של גוגל שהיא כבר הודיעה שבתחילת 2016 תבטל את התמיכה ב-SPDY בכרום, כדי להתיישר לסטנדרט.

וכך הגענו ל-HTTP2.

קדימה ישראל, תפניק אותנו ב-HTTP/2

בשמחה. מה שחשוב זה להבין את העקרונות והמטרות של HTTP/2 ואז נבין את כל מה שנעשה שם. העקרונות שעמדו בעת התכנון היו:

- להשאיר את מבנה הפרוטוקול של HTTP/1.1
- לשפר את הביצועים, כך שדף שעולה בדפדפן, יוכל להיטען מהר יותר
- ללמוד מטעויות העבר, בעיקר בתחום ה-Security

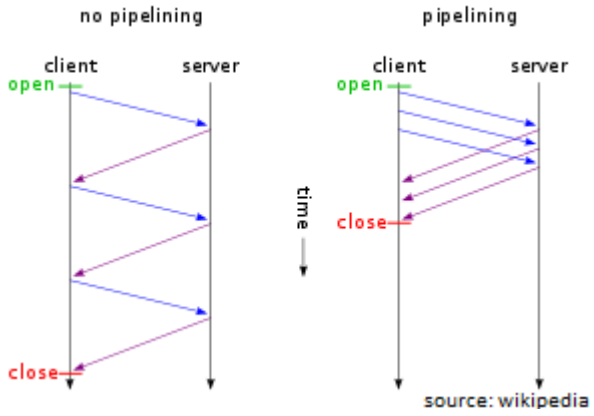
כדי לאפשר זאת, השתמשו במספר טכניקות, חלקם ברמה הטקסטואלית של הפרוטוקול וחלקם ברמת ה-Network.

להשאיר את מבנה הפרוטוקול של HTTP/1.1

ברמת האפליקציה, כל העיבוד של המידע, מבנה הבקשות, נותר זהה. כל מה שאנחנו מכירים על הכותרים (Headers), עוגיות (Cookies), מתודות (Get/Post ודומיו). גם הדחיסה הקלה שנראה בהמשך היא רק ברמת התעבורה, עבור הקוד בשרת האפליקציה הכל שקוף ונותר זהה.

Piplining over single TCP connection

בחיבור סינכרוני קלאסי, כנשלח בקשה (Request) - נקבל תשובה (Response). כשנשלח 2 בקשות, נקבל 2 תשובות. איך נדע איזו תשובה שייכת לאיזו בקשה? המודל הטבעי הוא FIFO. התשובה הראשונה שייכת לבקשה הראשונה. והתשובה השניה לבקשה השניה. מה שאומר שגם אם התשובה של הבקשה השניה אמורה הייתה להגיע מהר יותר, היא תחכה עד לתשובה של הבקשה הראשונה. ההמתנה מכונה HOL Blocking.



כדי להימנע מ"היתקעות" שכזו, צריך לעבור לתקשורת א-סינכרונית. הצורה הטבעית לעשות זאת היא להצמיד לכל בקשה איזשהו ID רץ. והשרת, כשהוא מחזיר תשובה מצמיד את ה-ID של הבקשה לתשובה. וכך גם אם נקבל את התשובה של הבקשה השניה קודם, נדע שהיא שייכת לבקשה השניה, לפי ה-ID. ב-HTTP זה מכונה http piplining. מדובר בהבדלי ביצועים משמעותיים מאוד.

המדהים זה שניתן לממש piplining גם על HTTP/1.1. ויותר מכך - מרבית הדפדפנים והשרתים תומכים בזה. אממה, כיוון שיש שרתים ש"התערבבו" מ-piplining, הדפדפנים מגיעים עם הפיצ'ר כשהוא Disabled.

בכל מקרה, ברגע שהחלטנו לממש piplining, אין עוד צורך בכמה חיבורים מול אותו שרת, אמור מעתה - HTTP/2 מבצע Single TCP connection מול השרת ומתקשר מעליו באמצעות ריבוב (Multiplexing). מבחינת HTTP/2, מסתכלים על הבקשה כמכילה 2 פריימים, פריים של ה-Headers ופריים של ה-Content. כך בבקשה וכך בתשובה, הפריימים רצים על החיבור בצורה שוטפת. המשמעות של הריבוב, היא שמספר פריימים יכולים לעבור (כמעט) באותו זמן והם לא מתבלבלים או מפריעים אחד לשני.

HPACK - Header compression

כדי לייעל את התעבורה, היינו מצפים שהדיפולט יהיה איזשהו כיווץ לתוכן הפריימים. הן על התוכן (מה שהיה מקובל לבצע באמצעות gzip) והן על ה-Headers. אך היות ובשנים האחרונות נפרצו מספר פרוטוקולים (ביניהם SPDY) שביצעו דחיסה למידע, גם כאשר התעבורה הייתה מוצפנת (ע"ע Crime, Breach) הוחלט לא לבצע דחיסה אלא אינדוקס.

קיימות 2 טבלאות. טבלה אחת סטטית, לכל מיני חלקים בפרוטוקול שהם נפוצים (Method GET, Method POST, Status code 200, Status code 404 ועוד). וטבלה אחת דינמית, שניתנת להגדרה פר קונקשן (כזכור, כל התקשורת של חלון הדפדפן עם שרת מסויים היא בחיבור יחיד).

תקשורת בינארית

אמנם התוכן הטקסטואלי של HTTP לא עובר המרה לבינארי, אבל כל יתר המידע של HTTP2 (טבלאות hpack, priority של פריימים וכד') עובר המרה. וגם התוכן הטקסטואלי עובר אנקפסולציה לתוך פריימים שהם מעטפת בינארית.

הסיבות העיקריות לשימוש בפרוטוקול בינארי הן שהוא קצר יותר. שזה אחד מהיעדים של HTTP2. והסיבה והסיבה השניה היא שהוא קל יותר לניתוח. הוא מוגדר מאוד ואין בו כל מיני שטיקים שקיימים בפרוטוקול טקסטואלי (ירידת שורה זה \n או \r\n. רווח ניתן לייצוג גם עם רווח, גם עם פלוס וגם עם %20 ודומיהם).

TLS also on HTTP

זה דבר שלא נקבע ברמת הפרוטוקול, אבל קרה דה-פקטו. פיירפוקס וכרום תומכים ב-HTTP2 רק מעל TLS, כך שגם אם בשורת הכתובת כתוב HTTP:// עדיין התקשורת תהיה מוצפנת. אם שני הדפדפנים הללו דורשים תקשורת מוצפנת, ההצפנה נהפכת להכרחית De Facto, ואין לשרתים טעם לתמוך ב-HTTP2 ללא TLS.

שנים שאני מחכה שיגיע הרגע הזה, והנה הוא בא. סוף סוף כל האתרים יהיו בתווך מוצפן, ומתקפות MITM יצטרכו להיות הרבה יותר חכמות.

Server side push

זה פיצ'ר שהוא בהחלט שונה ומעיד על חשיבה מקורית. הרי אנחנו בשרת, יודעים איזה קבצים נדרשים עבור הדפדפן לטעון את הדף. אז למה שנוריד לדפדפן רשימה של Resources ואז נחכה לקבל ממנו בקשות? אפשר במקביל, יחד עם רשימת ה-Resources להזרים עוד קבצים/Responses מהשרת. הדפדפן יכניס את הקבצים לקאש (מטמון), וכשיצטרך יטען אותם משם. דוגמת קוד Server side ב-NodeJS ש"דוחפת" קבצים לקליינט, ניתן לראות כאן.

מבחינת אבטחה, מיד קופצת לנו המחשבה על העמסה של הקאש של הדפדפן. אך זו בעיה שקיימת בלי קשר לשאלה אם הטריגר להורדת הקבצים בא מהדפדפן בעקבות דף html שהוא קיבל מהשרת או שהשרת שלח מיד לדפדפן את כל הקבצים. בכל מקרה הדפדפן צריך לאכוף איזושהי הגבלה.

לקריאה נוספת בעניין הפרוטוקול

וודאי יש לכם בראש שאלות כמו למה הוא נקרא HTTP/2 ולא HTTP/2.0? האם חושבים כבר על HTTP/3? האם זה ייתמך בקליינטים שהם לא דפדפנים? וכל מיני נקודות נוספות, אם כן כדאי לכם לקרוא את ה-FAQ הקליל של http2.github.io.

איזה שרתי Web תומכים ב-HTTP/2?

במפתיע, מרבית השרתים בגרסאות עדכניות כבר תומכים. IIS, Apache, Nginx, Jetty ועוד. חלקם בצורה מלאה, חלקם ע"י הפעלה של מודול וחלקם ברמת Experiment (רשימה מלאה). כמו שניתן לצפות, נראה שרתי פרוקסי/CDN-ים קדמיים (ע"ע אקאמאי) שתומכים HTTP2 ומאחורה שרתים שלאו דווקא תומכים.

תכל'ס, ישראל, בוא תראה לי איך זה נראה בפועל

כאנשי אבטחה וכמפתחים שמתעניינים באבטחה, מעבר לידע התיאורטי מעניין אותנו לנטר בפועל את התעבורה של HTTP2. יכול להיות שהדפדפן פונה ל-Endpoints אחרים באתר שלא הכרנו. אולי הם מכילים בעיות אבטחה שלא קיימות ב-Endpoints הישנים? אולי הוא בכלל שולח מידע לדומיין אחר?

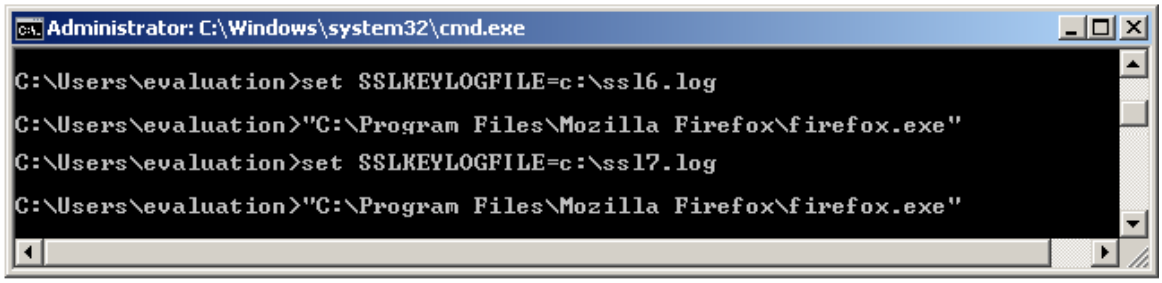
לכן מפתיע מאוד שכיום, כבר מספר חודשים אחרי שהפרוטוקול קיים בשטח, אתרים כמו גוגל ואקאמי משתמשים בו כבר (גם מתוך אתרים אחרים החיבור הוא ב-HTTP2). עדיין, אם תשימו Burp, Fiddler או כל כלי נפוץ אחר בין הדפדפן לשרת בניסיון לתפוס את התעבורה, זה לא יעבוד. הכלים הללו נכון לשעת כתיבת שורות אלו לא תומכים בפרוטוקול HTTP/2 (בפורום של Burp, למשל, נכתב ש[לא ידוע מתי הוא יתמוך]), וההתנהגות של האתר תהיה כמו בדפדפן שלא תומך HTTP2.

אז איך בכל זאת ניתן לראות את התעבורה? התשובה המפתיעה היא Wireshark. כאן אתם אמורים לקפוץ "הי, אבל אמרת שב-HTTP2 התעבורה תמיד מוצפנת!". אכן, היא מוצפנת, אבל פיצ'ר שמרבית האנשים לא מכירים מאפשר לפענח גם תעבורה מוצפנת דרך Wireshark. הביטו וראו.

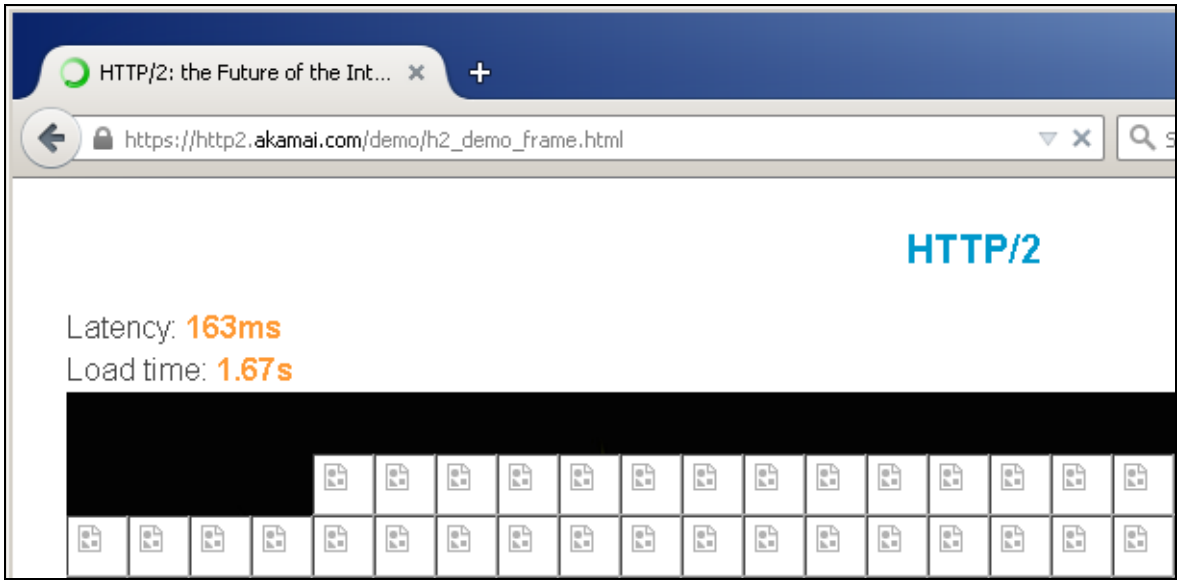
הערה: מכאן עד לסוף המאמר יהיו הרבה תצלומי-מסך, כדי שלא תבזבזו את השעות שאני השקעתי בחיפוש למה דברים לא עובדים לי...

דבר ראשון, אנחנו רוצים גרסת Wireshark שיודעת לנתח תעבורה של HTTP2, אז תורידו Wireshark בגרסת Development release. העדכני כיום (וממנו התצלומים) הוא 1.99.8. הוא נראה מעט שונה מהגרסה הרגילה. הורדנו? נפעיל אותו ונגיד לו להתחיל להאזין.

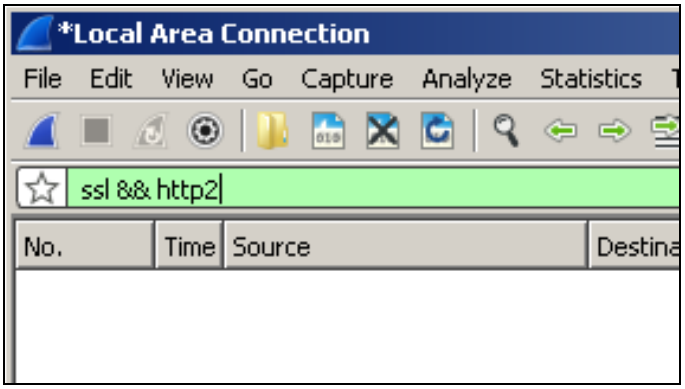
השלב הבא, יהיה להגיד לדפדפן להכניס לקובץ לוג מיוחד, את כל המידע הסודי שהוא מייצר בשלב יצירת חיבור מוצפן מול השרת, כדי ש-Wireshark יידע לקרוא אותו (לינק לפירוט טכני). נעשה זאת ע"י הגדרה של משתנה הסביבה SSLKEYLOGFILE לכתובת הקובץ (אין צורך ליצור את הקובץ מראש), ואז נפעיל את פיירפוקס/כרום.



בדפדפן שנפתח נגלוש לדף שמתקשר ב-HTTP2 (לינק):

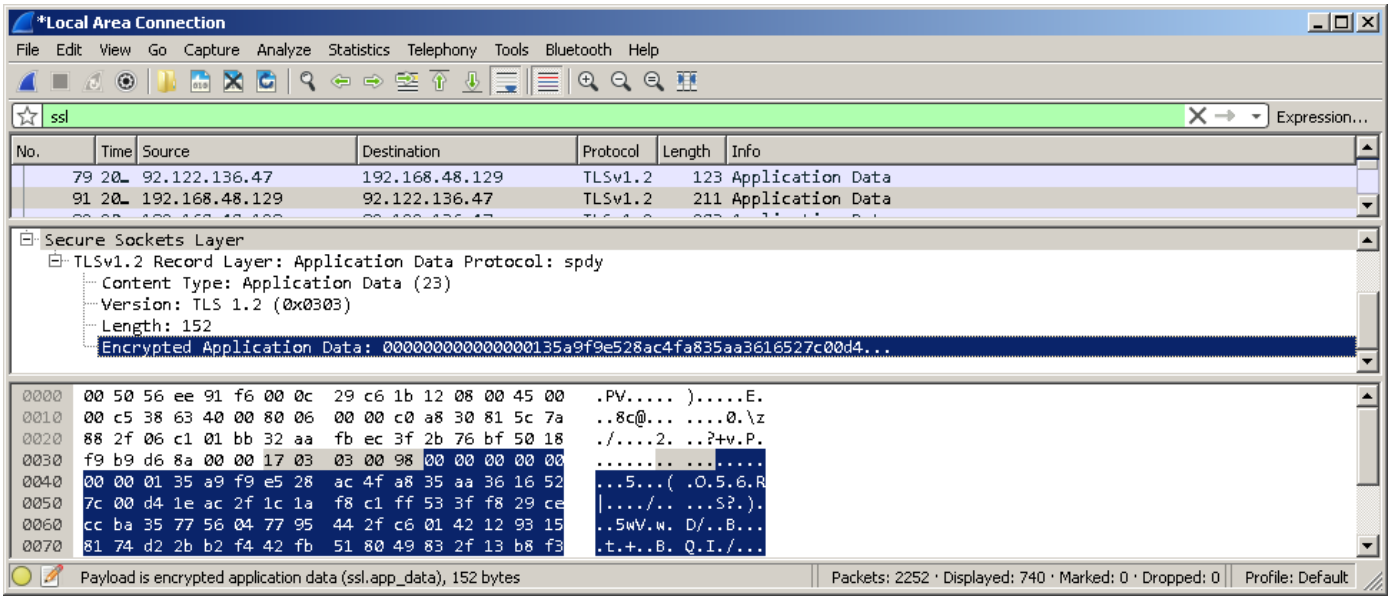


ב-Wireshark אנחנו אמורים לראות כעת תעבורה הולכת וגדלה. רק כדי "להוכיח" שהתעבורה של HTTP2 מוצפנת, נריץ פילטר על ssl && http2:



הסיבה שהתוצאה היא חלון ריק, היא שברגע שהתקשורת מוצפנת, ל-Wireshark אין אפשרות לדעת איזה פרוטוקול יש מעל ה-SSL.

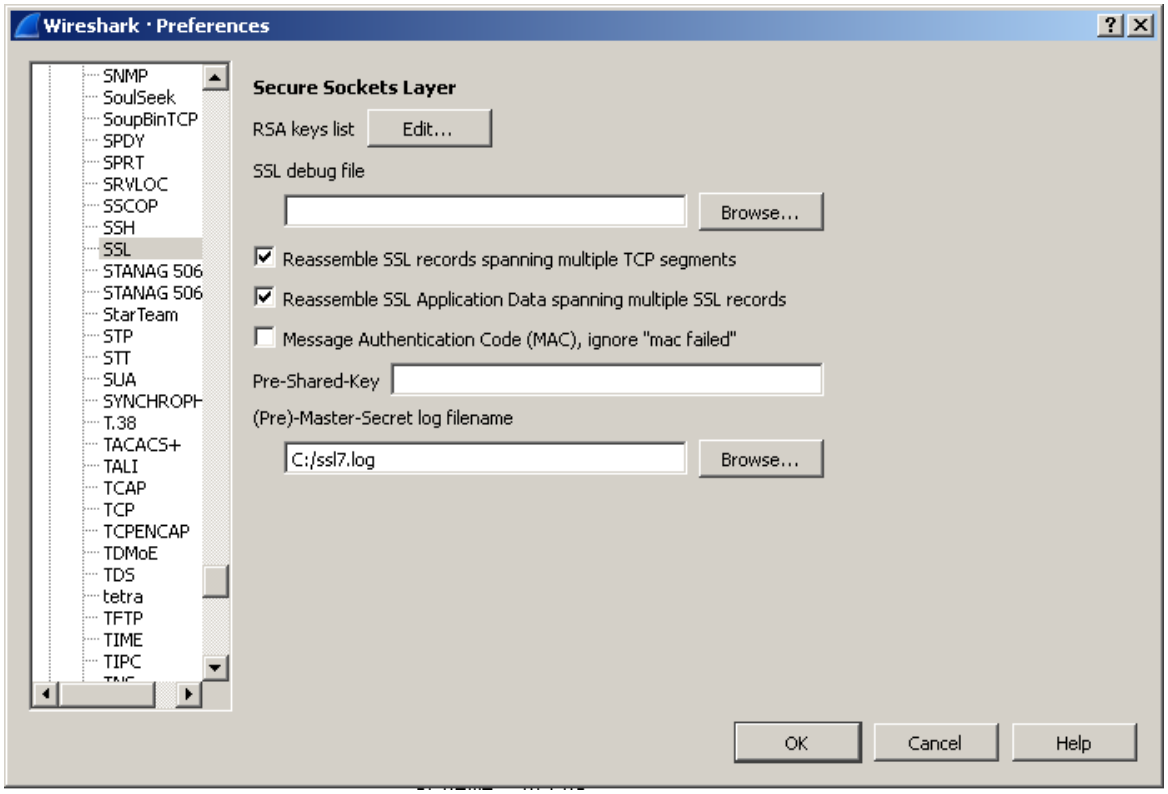
אפשר גם להריץ סינון על SSL. ללכת לאיזו שורה שה-Info שלה הוא Application Data, ולראות שהתוכן שלו הוא Encrypted application data:



הסיבה שאני מפרט את כל זה, היא שלפעמים בטסטים משהו לא עובד, ואתם לא יודעים אם הבעיה בפענוח, או שבכלל לא תפסתם טראפיק נכון, ואולי פיענחתם ואתם לא מבינים את זה... אז ככה תדעו להשוות באיזה שלב אתם נמצאים.

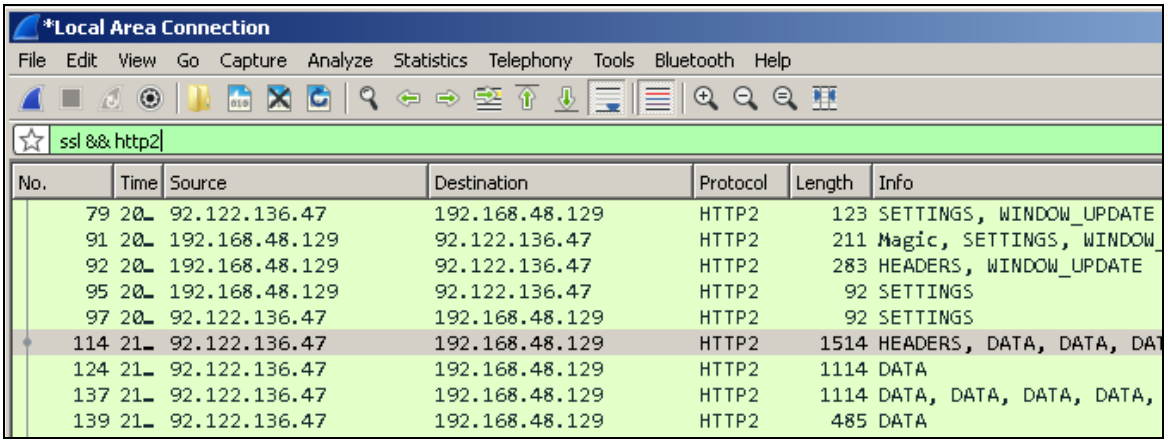
כעת ניכנס לתפריט Edit, ושם נבחר את הפריט התחתון ביותר, Preferences. בחלון שנפתח, במלבן השמאלי נפתח את Protocols, ובו נבחר את SSL.

כעת בצד ימין נגדיר את ה-Pre-Master-Secret log filename לקובץ שהגדרנו במשתנה SSLKEYLOGFILE:



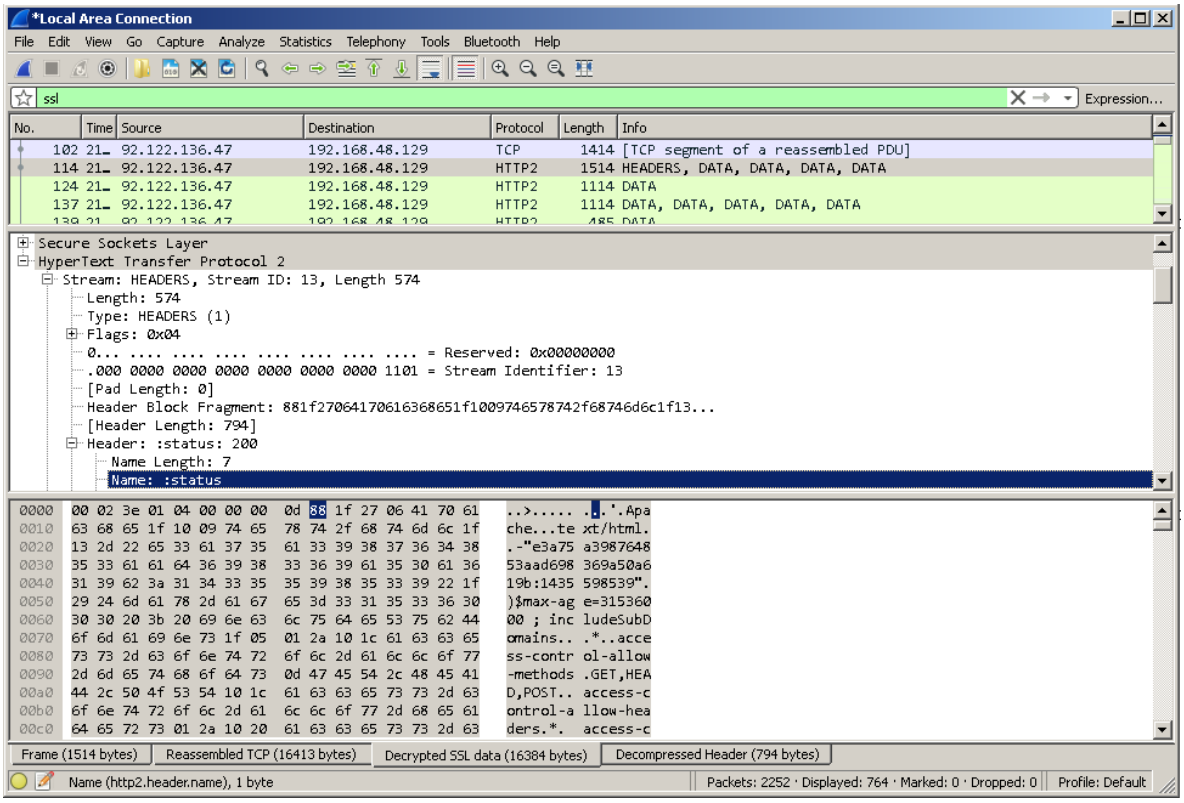
נלחץ OK, וזהו. כעת הכל אמור להיות מפוענח.

נריץ שוב את הפילטר של קודם ssl && http2 והפעם נקבל תוצאות:



כמו שאפשר לראות, בעמודת ה-Info מעבר ל-Headers ו-Data, יש עוד דברים שעוברים בתחילת החיבור. זוכרים את ה-Hpack שמאפשר להעביר טבלה דינמית? אז ככה יש כמה דברים שעוברים, ואז מתחיל לעבור המידע עצמו ב-2 סוגי Frames. אחד של Headers ואחד של ה-Data. בדיוק כמו שקראתם מקודם.

כעת נבחר שורה שמכילה Headers ונעיף בתוכה מבט:



הדבר הראשון שכדאי לראות זה שלמטה ממש מעל שורת ה-Status bar, יש טאבים תחתיים. הטאבים הללו לא היו לפני הפענוח של התעבורה. תחזרו שני דפים אחורנית לתצלום של Wireshark ותראו שאין שם את הטאבים.

שמתי את הפוקוס על הטאב Decrypted SSL data, ניתן לראות במלבן האמצעי שמדגיש בכחול את הטקסט: Name: :status וכפי שאפשר לראות 2 שורות מעליו מדובר בקוד 200. בכמה בתים לדעתכם נראה את זה במלבן התחתון? כפי שניתן לראות זה מיוצג בבית אחד. זו הדגמה חיה ל-Hpack שהזכרנו. 4 בתים אחריו ניתן לראות את ההידר Apache. איפה שם ההידר (Server)? גם הוא מועבר בבית יחיד. בדיקות מראות שהדחיסה הזו בהינתן מספר בקשות, חזקה הרבה יותר מ-Gzip!

נחזור למלבן האמצעי, שמתם לב לשורה המסומנת? Name: :status. מה זה הנקודתיים לפני המילה סטטוס? ובכן, ב-HTTP2 כשרואים את זה, סימן שהטקסט לא הגיעה כך אלא הגיעה באינדוקס Hpack

ומה שאנחנו רואים הוא הפענוח שלו. אם נרצה לראות את כל הבקשה במלבן התחתון מפוענחת בצורה חלקה, נלך לטאב Decompressed Header ושם נראה את הבקשה כמו בקשת HTTP רגילה.

מה שחסר שם זה הצלבה מלאה לעומת המלבן האמצעי. כשתשחקו עם זה טיפה תבינו וגם תלמדו עוד כמה דברים שאין טעם כעת להאריך בהם (לדוג' בפועל נעשה שימוש בפסאדו-הידר authority במקום בהידר המוכר host).

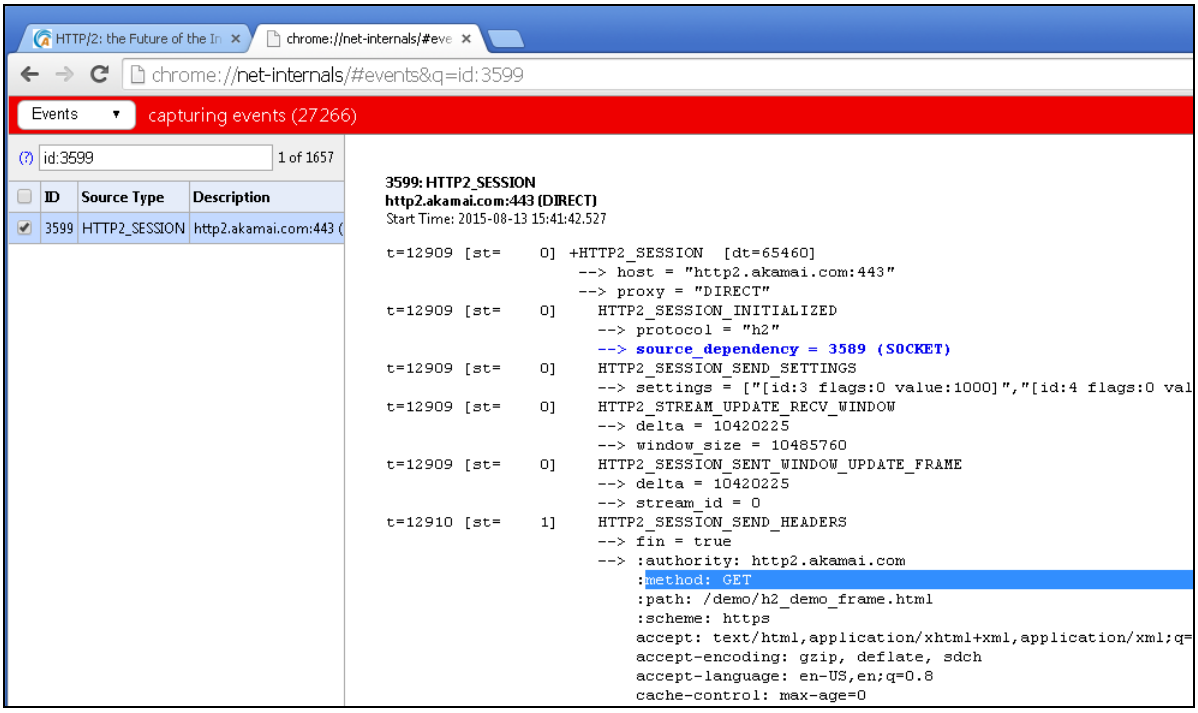
עד כאן בעניין Wireshark. אני רק אגיד ש-Wireshark יודע לפענח את המידע ה-SSLי ל-HTTP2 רק אם הוא היה בהאזנה (capture) מתחילת החיבור. כך שאם ביצעתם recapture או חלילה פתחתם את wireshark אחרי שגלשתם בדפדפן לכתובת היעד והדפדפן כבר התחבר פעם אחת לשרת. Wireshark לא יידע לפענח את התעבורה.

הצורה הקצרה לפתור את זה, לבצע capture ב-wireshark ואז לסגור ולפתוח מחדש את הדפדפן. אם רוצים לבצע "full restart", צריך לסגור את הדפדפן ו-Wireshark. לחזור לחלון CMD. להפנות את משתנה הסביבה SSLKEYLOGFILE לקובץ אחר (לא למחוק את הקובץ הקודם אחרת wireshark יקרוס כשהוא יתחיל להאזין, כי הוא עדיין מפנה לקובץ הישן), להפעיל שוב את Wireshark ולבצע Capture, להפעיל את הדפדפן מתוך ה-CMD שוב. וזהו.. לכן בתמונה הראשונה של חלון ה-CMD (3-4 דפים אחורנית) ניתן לראות שפעם הפניתי לקובץ ssl6.log ופעם לקובץ ssl7.log.

כלי נוסף שמאפשר לצפות בתעבורה, הוא לא אחר מאשר הדפדפן כרום בכבודו ובעצמו. מגיע איתו tool מובנה בשם net-internals שיושב בכתובת chrome://net-internals. אגב טיפ בעניין כרום, אם אתם לא זוכרים את הכתובות הפנימיות שלו, תמיד כשתתחילו לכתוב chrome:// אחת האופציות היא chrome-urls ששם יש את כל הכתובות.

נחזור לענייננו, גולשים ל-chrome://net-internals ולמעלה בצד שמאל בוחרים http2. כעת כשתגלשו מטאבים אחרים בכרום לאתר שמשתמש ב-http2 תראו "סשנים". בתמונה ניתן לראות שבחרתי סשן ונפתח חלון בצד ימין שמראה את התקשורת של הסשן.

בתחילה יש את החלקים ה-"Network"ים של HTTP2, למטה ניתן כבר לראות הידרים של בקשה שנשלחים לשרת. שימו לב ל-method: אתם כבר יודעים למה יש שם נקודתיים...



לגבי עריכה של הטראפיק, לצערי כרגע אין כלי שמאפשר את זה, אז תצטרכו לבצע קומבינות (תוסף לדפדפן שמבצע hooks לפונקציות JS שמוציאות את הבקשות, תוסף ל-Wireshark, או כל דבר שאתם חושבים עליו).

אפילוג

ההרצאה הסתיימה, סשן קצר של שאלות ותשובות, ותוך כדי שהמשתתפים מגלגלים במוחם רעיונות לתקיפה של הפרוטוקול, טכניקות הגנה וכל מיני דברים שהאקרים חושבים עליהם, המרצה הבא נעמד ומחבר את המחשב שלו למקרן. טק-טוק מכיל בדרך כלל 2-4 סשנים. בהמשך גם הוצגה מתקפה שמאפשרת Code execution בשרתים. כאמור, שעה של נחת.

על הכותב ומשפחת אפסק

ישראל חורז'בסקי, CTO באפסק. בין השאר גם מרצה ויועץ אבטחת מידע בתחום Application Security.

החדשות הטובות הן - אפסק מגייסת עובדים! הן Juniors והן Experts. יש 2 סיבות למה לעבוד באפסק: 1. זו חברה שנמצאת בטופ של הטכנולוגיה ושל המתקפות האפליקטיביות, ומשקיעה כל העת כדי להישאר שם. Period! 2. היחס לעובדים הוא אישי, עושים הכל כדי לבוא לקראתם, אנחנו משקיעים בעובדים, נותנים ואפי' דורשים כל העת להתקדם.

אז אם אתה רוצה להתקדם - מקומך איתנו, שלח קו"ח ל-israel@appsec-labs.com.

שלכם

ישראל חורז'בסקי

סמנכ"ל טכנולוגיות (CTO), AppSec Labs