digital whisper
Technologic Papers

ReDTunnel - Redefining DNS Rebinding Attack

מאת נימרוד לוי ותומר זית

הקדמה

האם חשבתם פעם כיצד ניתן לקבל גישה לרשת פנימית של קורבן מבלי להריץ על המחשב שלו Malware או להשתמש ב-0Day? באמצעות גלישה לאתר אינטרנט המכיל סקריפט מתוחכם אשר יספק לכם Tunnel מלא לרשת הפנימית של הקורבן? גם אנחנו וזו הסיבה שהחלטנו לכתוב את הכלי **ReDTunnel**.

המטרה היא להשתמש בכלים הקיימים במערכת ההפעלה של הקורבן (דפדפן), על מנת לקבל גישה מלאה לרשת שלו, תוך מתן דגש להתחמקות ממערכות EDR ומבלי להעלות סימני שאלה במערכות הזיהוי הארגוניות (IPS).

החלטנו לשלב טכניקות Reconnaissance ב-JavaScript והתקפת DNS Rebinding, כדי להגשים את המטרות שלנו.

כל מה שעליכם לעשות הוא לגרום לקורבן לגלוש לאתר אינטרנט המכיל את הסקריפט שמחבר לסביבת הכלי (**ReDTunnel**), ומשם המשחק מתחיל! לאחר כשתי דקות, תוכל לגלוש באופן מוחלט לנכסי ה-Web הארגוניים (לדוגמא ראוטר, NAS וכו') מיותר לציין כי זו דרך מעולה להתחלה של מבדק Red Team מוצלח.

DNS Rebinding

הסבר מפורט על DNS Rebinding כבר נכתב בדיגיטל וויספר בעבר, אז במקום להסביר לכם שוב אנחנו ממליצים לכם לקרוא על DNS Rebinding במאמר של אביעד בגיליון 9.

המגבלות ב-DNS Rebinding

מתקפת DNS Rebinding דורשת המון אינפורציה על הקורבן כדי לעבוד, למשל צריך לדעת איזה ממשקים אנחנו תוקפים, למשל אם אנחנו תוקפים ראוטר אז צריך לדעת מה הכתובת אייפי של הראוטר ברשת של הקורבן ולאחר מכן איזה בקשות אנחנו צרכים לשלוח לראוטר בכדי להתחבר אליו בהצלחה או לתקוף אותו.

בימים שקדמו ל-**ReDTunnel** כל המידע הזה היה צריך להיאסף מבעוד מועד וכל תוקף נאלץ לכתוב בעצמו סקריפטים שישתמשו במידע שנאסף כדי לתקוף את הקורבן. כדי להמנע מהמגבלות הללו השתמשנו בטכניקות של JavaScript Reconnaissance.

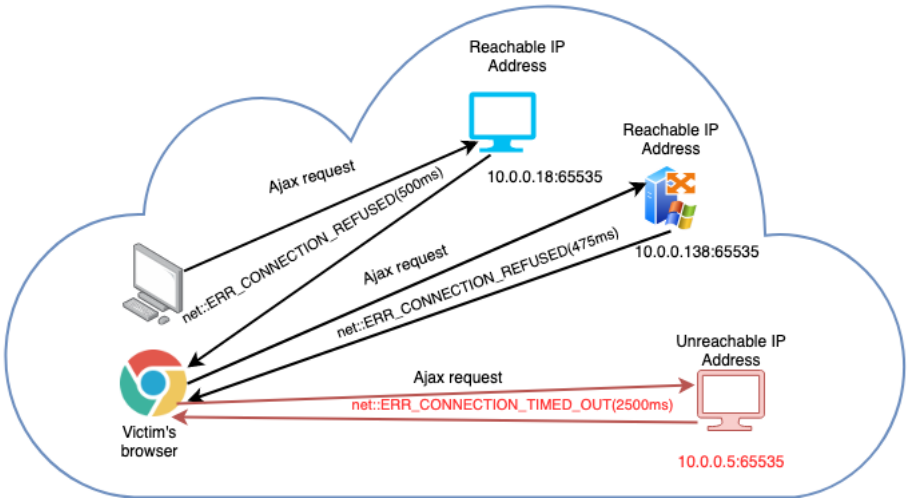
JavaScript Reconnaissance

Reconnaissance הינו תהליך איסוף המידע אודות הקורבן, לדוגמא באיזה דפדפן הקורבן משתמש? אילו הרחבות הקיימות בדפדפן שלו?, האם ניתן לקבל את כתובת ה-IP הפנימית שלו? איזה כתובות IP פתוחות אצלו ברשת? איזה ממשקים פתוחים אצלו ברשת וכו'

אתם תתפלאו כמה JavaScript יכול לעזור לנו במלכה הזאת וכמה מידע אפשר לדלות דרך סקריפט אחד פשוט....

אז איך מתבצע תהליך ה-Reconnaissance באמצעות JavaScript? ישנן המון שיטות לאיסוף מידע, אנו נתבסס על אחת מהן לדוגמא. כאשר אנו מבצעים פניה לכתובת IP קיימת עם פורט שאינו פתוח, נקבל שגיאה בפחות מ-2 שניות אודות כשלון ההתחברות (net::ERR_CONNECTION_REFUSED). כמו כן, נקבל את שגיאה שונה אודות כתובת IP שאינה קיימת לאחר יותר מ-2 שניות (net::ERR_CONNECTION_TIMED_OUT), את סוג השגיאה לא נוכל לבדוק ב-JavaScript, אך כן נוכל למדוד את הזמנים ולדעת שקרתה שגיאה כלשהי...

בואו ננסה להבין את המנגנון



בתמונה מצד שמאל ניתן לראות את הדפדפן של הקורבן אשר מבצע פניות לכתובות IP שונות, כאשר החצים האדומים מסמנים התקשרות לכתובת IP שאינה קיימת ברשת, ואילו החצים השחורים מסמנים כתובות IP אשר קיימות ברשת.

כאשר מתבצעת פניה לכתובת IP פנימית ברשת, הדפדפן מתשאל את הראוטר האם הוא מכיר את הכתובת, במידה והוא מכיר אותה הוא יחזיר תשובה בצורה מהירה על כך שהפורט סגור ולכן נקבל תשובה מהירה אודות הפורט הסגור ונקבל שגיאה אודות החיבור שנדחה - עד כאן ברור.

במידה וננסה לגשת לכתובת IP שאינה קיימת, הדפדפן יחכה ל-Timeout ועל פי כך נוכל לדעת בוודאות שכתובת ה-IP אינה קיימת.

חשוב לציין שלכל דפדפן ישנן מגבות משלו, למשל השיטה שציינו תעבוד על Google Chrome אך לא תעבוד ב-FireFox.

לעומת זאת ב-FireFox נוכל לשלוח הרבה יותר בקשות בו זמנית אז לא נצטרך להשתמש בשיטה הזאת כדי לדעת שאנחנו שולחים בקשות אך ורק לכתובות IP קיימות.

אז כמו שאמרנו כדי ש-ReDTunnel יצליח אנחנו צריכים לאסוף מידע על הקורבן, לכך השמשתנו במספר טכניקות JavaScript Reconnaissance שונות:

1. **גילוי כתובת האייפי הפנימית:** כדי לגלות את כתובת האייפי הפנימית השתמשנו ב-WebRTC, סקריפט לדוגמה:

```
function getLocalInterfaceIps() {
  return new Promise((resolve, reject) => {
    if (!window.RTCPeerConnection || !window.RTCPeerConnection.prototype.createDataChannel) {
      return reject(new Error('WebRTC not supported by browser'));
    }

    let pc = new RTCPeerConnection();
    let ips = [];

    pc.createDataChannel("");
    pc.createOffer()
      .then(offer => pc.setLocalDescription(offer))
      .catch(err => reject(err));
    pc.onicecandidate = (event) => {
      if (!event || !event.candidate) {
        // All ICE candidates have been sent.
        if (ips.length === 0) {
          return reject(new Error('WebRTC disabled or restricted by browser'));
        }
        return resolve(ips);
      }

      let parts = event.candidate.candidate.split(' ');
      let [base, componentId, protocol, priority, ip, port, , type, ...attr] = parts;

      if (!ips.some(e => e === ip))
        ips.push(ip);
    };
  });
}
```

2. **גילוי כתובות אייפ קיימות ברשת:** כדי לגלות את כתובות האייפי הקיימות ברשת השמתשנו בתזמונים לבקשות Ajax הסבר על כך ניתן לראות בדוגמה של JavaScript Reconnaissance. סקריפט לדוגמה ניתן למצוא ב:

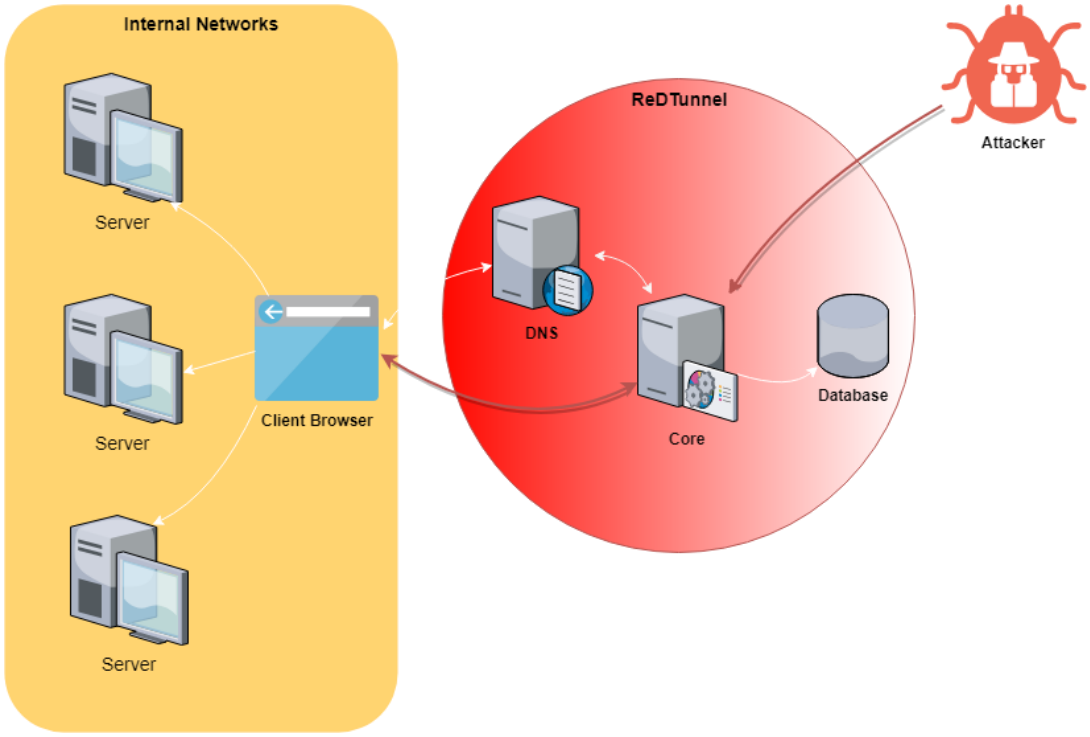
https://github.com/ReDTunnel/redtunnel-core/blob/master/views/ping.ejs

3. **גילוי פורטים עם שירותי HTTP:** כדי למצוא פורטים עם שירותי HTTP יצרנו אלמנט Script ב-DOM שה-Source שלו מצביע לפורט שאותו אנחנו בודקים, לפי התזמונים והאיוונטים שקופצים נדע אם מאחורי הפורט הזה באמת יש שירות HTTP, האיבנט שאנחנו צריכים הוא onload אם onload נקרא יש מאחורי הפורט שירות HTTP.

סקריפט לדוגמה ניתן למצוא ב:

https://github.com/ReDTunnel/redtunnel-core/blob/master/views/scan.ejs

עכשיו נביט בארכיטקטורה של הכלי ונסביר את התהליכים שקורים מאחורי הקלעים:



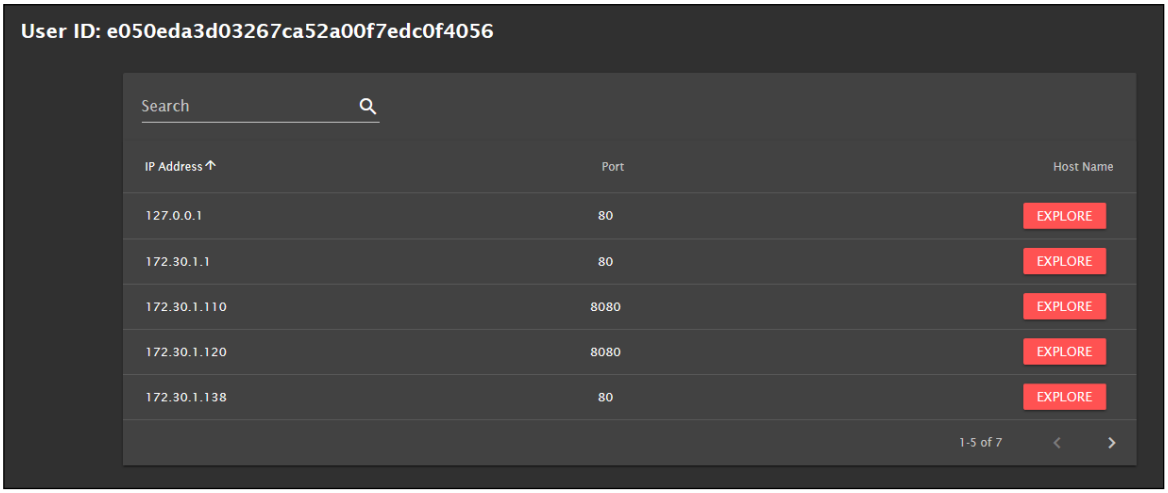
1. הקורבן גולש לאתר כלשהו שמכיל את הדף של ReDTunnel (Core). שירות ה-Core של ReDTunnel מכיל גם את צד התקיפה וגם את צד הניהול.

2. ReDTunnel אוסף מידע על הקורבן: מציאת כתובתו הפנימית של הקורבן, שימוש בכתובת הפנימית לסריקת כתובות ה-IP ברשת הקורבן ולאחר מכן מציאת שירותי HTTP ברשת הקורבן על כתובות ה-IP שנמצאו. וכמובן דיווח על כל הממצאים לשרת הניהול של ReDTunnel (Core).

```
Console:
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [172.30.1.110:8080]
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [127.0.0.1:80]
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [172.30.1.3:80]
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [172.30.1.120:8080]
• [17:32:14.349] - User: e050eda3d03267ca52a00f7edc0f4056 - ping - [172.30.1.2]
• [17:32:14.349] - User: e050eda3d03267ca52a00f7edc0f4056 - ping - [172.30.1.1]
• [17:32:14.349] - User: e050eda3d03267ca52a00f7edc0f4056 - ping - [172.30.1.138]
```

3. לאחר איסוף המידע התקפת ה-DNS Rebinding מתבצעת בצורה אוטומטית (באמצעות תקשורת בין שירות ה-Core לשירות ה-DNS).

4. התוקף משתמש בממשק ה-Admin כדי לגלוש לאפליקציות ברשת של הקורבן או על מחשב הקורבן (localhost) באמצעות הדפדפן ואף יכול להפעיל סקריפטים אוטומטיים כמו SQLMAP (אם ישלח את תוכן ה-Cookie של redtunnel).



לסיכום

הכלי ReDTunnel הגדיר מחדש את מתקפת ה-DNS Rebinding על ידי אוטומציה של המתקפה, איסוף המידע לפני המתקפה וניהול הקורבנות בצורה בטוחה ומתוחכמת (מאפשר לגלוש פיזית ברשת הפנימית של הקורבנות ולהפעיל עליהם כלים אוטומטיים, כמו כן מאפשר מטודות כמו DELETE / PUT / PATCH וחלחול התחברות ה-Basic Authentication לתוקף.

קישורים בנושא

- https://www.digitalwhisper.co.il/files/Zines/0x09/DW9-3-DNSRebind.pdf
- https://github.com/ReDTunnel
- https://www.blackhat.com/asia-19/arsenal/schedule/#redtunnel-explore-internal-networks-via-dns-rebinding-tunnel-14332
- https://portswigger.net/daily-swig/new-tool-enables-dns-rebinding-tunnel-attacks-without-reconnaissance

תודות

- למקס רינקה (muhaack - מקעקע ואמן גרפיטי) על הלוגו המטורף.
- לדימה בלסקי (מתכנת Full Stack ב-F5 Networks) על ה-UI המגניב.

על המחברים

- **תומר זית (RealGame):** חוקר אבטחת מידע בחברת F5 Networks וכותב Open Source.
  - אתר אינטרנט: http://www.RealGame.co.il
  - אימייל: realgam3@gmail.com
  - GitHub: https://github.com/realgam3
- **נימרוד לוי (El3ct71k):** חוקר אבטחת מידע, CTO בחברת Scorpiones ומפתח Open Source בזמנו הפנוי.
  - אימייל: El3ct71k@Gmail.com
  - GitHub: https://github.com/El3ct71k