VB2020 localhost

30 September - 2 October, 2020 / vblocalhost.com

THE OST MAP: MAPPING THE USE OF OPEN-SOURCE OFFENSIVE SECURITY LIBRARIES IN MALWARE

Paul Litvak

Intezer, Israel

litvakpolka@gmail.com

ABSTRACT

The unrestricted publication of offensive security tools (OSTs) has become one of the most controversial talking points in the information security community. Some argue that releasing such tools to the Internet is irresponsible, as it allows adversaries to outsource the development of tools and techniques from the infosec community directly. Others believe the publication of these tools serves as a cornerstone to the education of both new researchers and good defence practices, allowing defenders to mitigate newly discovered techniques and probe their own systems. However, limited quantifiable data has been presented to support either argument.

This research was conducted in order to evaluate the extent of influence offensive security tools have on adversary operations, specifically the use of open-source OSTs. We gathered leading open-source projects such as Mimikatz and UACME and compiled them with various configurations and flags in order to generate all possible binary code patterns. We identified code reuse patterns across a database of millions of malware samples and created a map of open-source OST adoption by malware families.

In this paper, a comprehensive map of the relationship between various OST open-source projects and threat actors is presented, i.e. the use of code injection, privilege escalation, and lateral movement technique implementation projects. We also explain the steps taken to build the map. Finally, we explain how familiarity with these projects allows defenders to build YARA signatures based on code patterns and expose real, undetected malware campaigns that were discovered based on this technique, together with the relevant YARA signatures.

INTRODUCTION

The term 'offensive security tool' is often misunderstood and is defined loosely as any tool that promotes adversarial operations. We have chosen to present a more exact definition, written by one of the leading voices against the uncontrolled proliferation of OSTs: 'Offensive Security Tools are the aggregation of disparate functionality combined and streamlined to facilitate authorized intrusions or to circumvent existing security measures without leveraging a software bug' – Andrew Thompson, Manager, Advanced Practices at *FireEye* [1].

It's important to distinguish between tools that are specifically designed for offensive operations and tools that are merely abused for their conduct. An example is any *SysInternals* tool, such as *PSExec* or *ProcDump*, which are often abused by threat groups. These tools should not be labelled as offensive tools because their purpose is to promote productivity rather than provide offensive functionalities. We believe OSTs should be labelled by the infosec community based on assessment of said tools' usefulness as opposed to the damage inherent in their free distribution to adversaries.

Adversaries with all types of sophistication levels use OST tools. From ransomware groups to top government agencies, Mimikatz has been deployed in many operations. Many of these tools are shipped as independent executables and are used as is with few modifications.

Due to the infosec community's familiarity with offensive tools, it's easy to detect their use and further quantify their influence on threat actors.

As more threat actors have outsourced the creation of capabilities to open source tools, some have started questioning whether the overall benefit of the unrestricted publication of these projects is positive. This side believes that the current climate, in which information security professionals publish offensive tools to the public, helps arm adversaries and misses opportunities to exact costs from adversaries. Many from this camp believe the publication of such tools should move into a closed forum of security professionals to which access would be granted only to trusted members, so capabilities are not able to fall into the wrong hands.

Others believe it's important for OST projects to be released publicly. They claim these tools improve the overall security situation by educating defenders and supplying them with practical tools to probe and strengthen their own defences. Furthermore, they believe the exclusivity solution proposed by the other side would be impossible to maintain or would impose a financial burden for entry which would particularly hurt newcomers from poor countries.

In order to add more substance to the debate, we were interested in quantifying the influence of these types of tools and their incorporation by threat actors. Furthermore, we were interested in code that was harder to detect – code with offensive capabilities that was embedded into custom tool sets by threat actors, and is not usually reported by vendors.

For this, we examined the effect of libraries that provide offensive security capabilities, or strips of code taken from larger 'as-is' tools (such as Mimikatz and Metasploit), that are incorporated into malware.

Can such libraries be labelled as OSTs in the same light as dedicated tools and frameworks? Unlike dedicated tools and frameworks, libraries impose higher barriers of entry for users. Libraries are often meaningless when deployed independently and therefore require integration efforts into existing toolsets. We hope to help the reader make up his mind on this matter by the end of the paper.

FINDING THE CONNECTIONS

Our research focused on native tools/libraries due to our expertise in detecting connections between native artifacts. Script-based OST projects were also incorporated into the connections map, though only based on current vendor reports.

It was important for us to discover connections between offensive libraries and families without having to analyse each sample manually. One possible option was to build YARA rules for each library, however this would be time-consuming, especially if we were to include binary patterns into these rules.

Therefore, to identify the connections at scale we decided to use an in-house code reuse engine. Libraries in our system were dissected into assembly blocks and compared against our collection of known malicious samples. Each connection generated was verified manually to prevent false positives.

The main challenge we encountered using our code reuse approach was that different versions of Visual Studio compilers were generating different code for the same code base.

Therefore, we built a PowerShell wrapper [2], which we released in *GitHub*, and which was used to compile each library with multiple Visual Studio compiler backends in order to support a larger variety of code. We added support for Visual Studio compilers from VS2019 to VS2010. Older compilers demanded backwards-compatibility work with the code base, which was unscalable.

To ensure full coverage of all code related to each library, projects based on Git were polled for historic monthly commits to cover code that was deprecated as the project matured.

Finally, connections were enriched by scraping security vendors' reports for mentions of offensive projects. This was especially effective when samples were unpublished.

OST MAP

Overall, 80 projects were checked for code reuse against a database of thousands of labelled threat actor samples from multiple vendor reports from the last few years. A total of 29 additional script-based tools were added using existing vendor reports.

The map consists of open-source projects marked in blue, minor actors marked in yellow (e.g. multiple low-profile ransomware groups), sophisticated and well-known threats marked in orange (e.g. Carbanak), and government-sponsored groups marked in red. It is available at https://www.intezer.com/ost-map/.

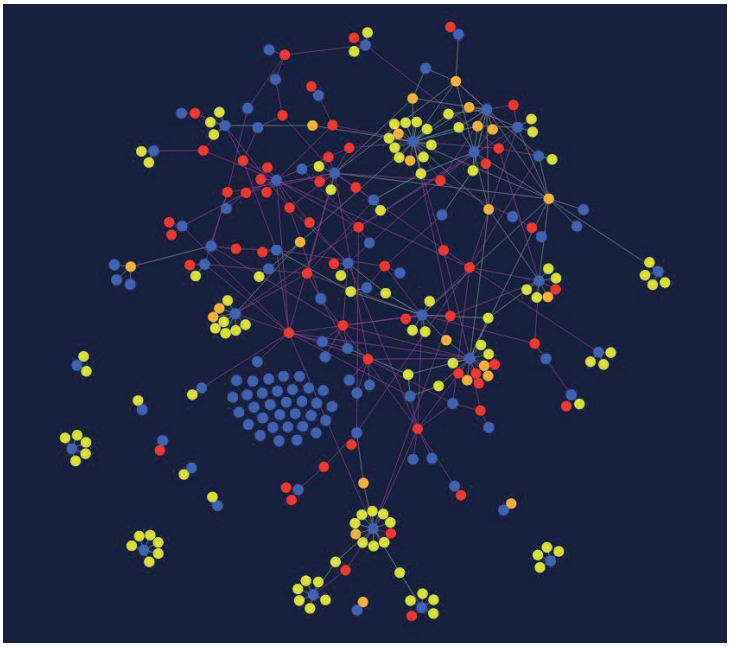

Figure 1: OST map.

In our map, we avoided adding unlabelled malware, which would clutter the graph with unrecognized nodes. It's important to note that some unused libraries are actually used by threat actors but these samples have never been published by security vendors.

While, due to the aforementioned constraint, the map doesn't paint the full picture of how well adopted certain tools/libraries are by threat actors, it's a valuable proxy to begin discussion about publication of offensive capabilities and to promote documentation of such usage by threat actors in future reports.

To understand which types of capabilities are more popular among threat actors, we grouped the tools by categories: memory injection libraries (20), RATs (including post exploitation tools) (23), credential access libraries (e.g. browser password stealers) (12), UAC bypass libraries (9), lateral movement libraries (including *Windows* password stealers used for lateral movement) (13), privilege escalation (5), utilities (11) and others (16).

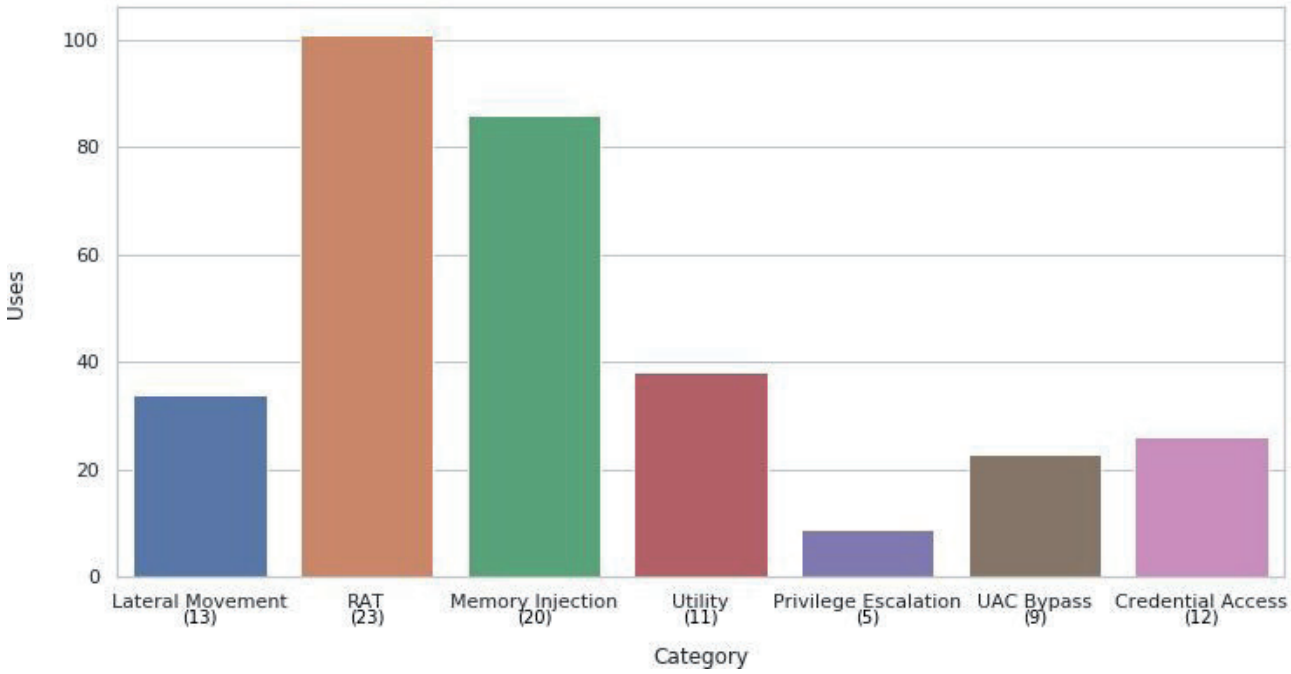

Figure 2: Tools grouped by category.

We found the most commonly adopted projects were memory injection libraries and RAT tools. The most popular memory injection tool was the ReflectiveDllInjection library, followed by the MemoryModule library. For RATs, Empire, Powersploit and Quasar were the leading projects.

The lateral movement category was dominated by Mimikatz in spite of the fact that only custom Mimikatz implants were included in the map, which reduced the overall number of connections for this library.

UAC bypass libraries were mostly dominated by the UACME library (a comprehensive compilation of known UAC bypasses). Interestingly, a small cluster of Asian-based threat groups were also using a lesser-known library, 'Win7Elevate' [3].

For credential access libraries, there was no clear leader. We discovered these libraries weren't as popular as other categories, despite many malware implants incorporating credential stealing features. We believe this is a result of their competitive disadvantage against more advanced stealers, originating from hacking forums. Hacking forums are rife with credential stealing tools for various types of software, resulting in a more feature-rich ecosystem than what open-source projects currently offer.

Another interesting observation we made is that there were many libraries with clear, advertised offensive capabilities that were seldom used by threat actors, despite attaining hundreds of *GitHub* stars. We noticed that this was particularly the case for libraries shipped with 'complex' features – capabilities that required a deeper understanding to use. Examples include TokenVator (privilege escalation), NetRipper (post exploitation), SharpSuite (memory injection) and HookPasswordChange (credential access).

Incidentally, many of these projects seem to possess the highest educational value, having also been documented well and often presented at conferences. Threat actors that did incorporate capabilities from such libraries were mostly adopted by sophisticated threat actors. For example, NetRipper, a network traffic interception tool, is only documented to have been used by a Russian government-sponsored APT [4].

Another indicator for unpopular libraries among threat actors was their publication date. Newer libraries received much less attention from threat actors when compared to older libraries, especially in saturated fields such as memory injection where newer libraries do not introduce novel improvements. Adoption of newer offensive libraries is slow.

Consider the pinjectra library, which has ~400 stars at the time of writing this paper. The library has been hosted in *GitHub* since mid 2019, however we did not detect any samples in our collection utilizing this library, nor has any vendor reported such usage.

Figure 3: pinjectra.

Overall, we see libraries and tools with the least integration hurdles, verbose documentation, and therefore lowest barrier of entry, as the most appreciated by threat actors.

TURNING OPEN SOURCE AGAINST THREAT ACTORS

Familiarizing ourselves with which libraries threat actors use allows us theoretically to know how parts of their code look. Code copied from libraries usually doesn't deviate much from its source, and we expect to recognize its patterns and flows once we come to understand it.

Consider code from the MemoryModule memory injection library – it's the only memory injection library that calls the SetLastError [5]API after validating PE headers for the image it intends to inject.

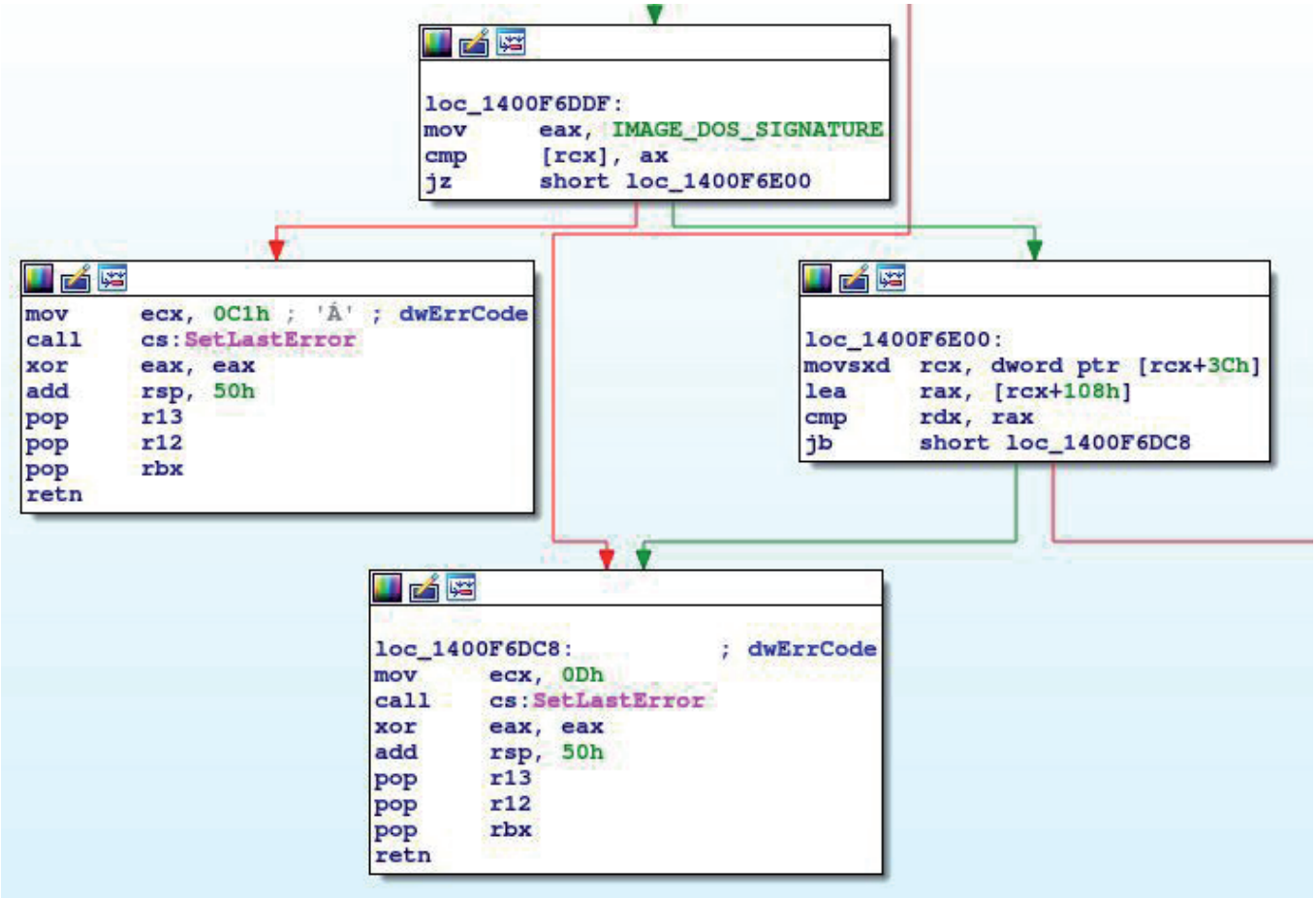

Figure 4: Code from the MemoryModule memory injection library.

Our knowledge of what such a code base looks like it is only available to us while reversing each sample separately. We wanted to find a way to transfer this knowledge of what such code looks like into a textual detection rule so we could find new samples.

For this purpose, we developed an in-house tool to cluster hundreds of samples compiled from the same project. We collected repeating binary patterns and automatically generated YARA rules based on binary patterns which are now available in our *GitHub* [6] repository for detecting possible threats.

```
private rule id_1
{
    meta:
        author = "Intezer Labs"
    strings:
        $a0 = { 8B [2] 8B [1] 89 [2] 85 [1] 0F 85 [4] 8B [2] 6A [1] 68 [4] 8B [2]
        $a1 = { 8B [2] 8B [1] 89 [2] 85 [1] 0F 85 [4] 8B [2] 6A [1] 68 [4] 8B [2]

    condition:
        any of them
}
```

Figure 5: YARA for the ImprovedReflectiveDLLInjection library.

We deployed these rules in *VirusTotal* and were able to find many undetected samples of interesting groups. For example, we were able to find almost undetected Lazarus [7] samples by tracking usage of the ImprovedReflectiveDLLInjection [8] library, and were able to identify the first instance of the adoption of this library by the group [9].

CONCLUSIONS

In this paper, we have reviewed the adoption of libraries and tools with offensive capabilities by known threat groups and presented data that we hope will promote a more fruitful debate regarding the considerations of publishing offensive security tools.

Our suggestion to tool authors wanting to limit their usefulness to adversaries is to raise the barrier of entry by requiring a deeper understanding of the techniques in order to use the library. This helps increase integration investment costs, while still retaining the libraries' educational value.

Another suggestion is to 'sprinkle' the library with special or irregular values in order to help defenders build signatures to defend against their use. For example, such an approach was adopted by the author of Mimikatz, where a generated ticket's lifetime is left to 10 years by default – a highly irregular number [10].

We invite the community to help enrich our understanding of threat actors' interaction with open source offensive security tools by documenting new OST connections in our *GitHub* [6] repository, which is used as the data source for the OST map.

REFERENCES

[1] https://medium.com/@QW5kcmV3/misconceptions-unrestricted-release-of-offensive-security-tools-789299c72afe.

[2] https://github.com/intezer/ost-map/tree/master/vs-autocompiler.

[3] https://www.pretentiousname.com/misc/win7_uac_whitelist2.html.

[4] https://download.bitdefender.com/resources/files/News/CaseStudies/study/170/Bitdefender-Whitepaper-Pacifier2-A4-en-EN.pdf.

[5] https://github.com/fancycode/MemoryModule/blob/master/MemoryModule.c#L572.

[6] https://github.com/intezer/ost-map.

[7] https://www.virustotal.com/gui/file/1e34709734b401413cc38818c1d7e34126fdc01a9bc47a1607e1371dd8d1385b/detection/f-1e34709734b401413cc38818c1d7e34126fdc01a9bc47a1607e1371dd8d1385b-1590555936.

[8] https://github.com/dismantl/ImprovedReflectiveDLLInjection.

[9] https://twitter.com/polarply/status/1270630283088461824.

[10] #StateOfTheHack: #DerbyCon Talks with @Carlos_Perez & @gentilkiwi. https://www.pscp.tv/w/1djxXRjdYAPGZ.

APPENDIX A – LIBRARIES/TOOLS LIST

Blackbone, SharpWeb, Tokenvator, HookPasswordChange, injectAllTheThings, coilgun, MemoryModule, ReflectiveDLLInjection, InjectProc, process-inject, SharpSploit, TikiTorch, LethalHTA, SharpExec, RottenPotato, JuicyPotato, SharpCOM, ProcessInjection, UACME, Trebuchet, .NetPELoader, Custom, Mimikatz, passcat, csharp-uhwid, minhook, mhook, DetoursNT, EasyHook, RemCom, BypassUAC/Win7Elevate, unknown_reflective_injector, K8-BypassUAC, PSInject, QuasarRAT, HTran, QuarksPwDump, ImprovedReflectiveDLLInjection, fgdump, WinEggDrop, PortScanner, cardmagic, SimplePELoader, ProxyDll, Dumpert, RunPE-In-Memory, Sharpire, Virtual, Machines, Detection,

Enhaced, rewolf-wow64ext, Invoke-Vnc, Covenant, SharpSuite, Process-Hollowing, TpmInitUACBypass, tenable-UACBypass, DccwBypassUAC, ALPC-BypassUAC, WheresMyImplant, SharpHound, NetRipper, InfinityRAT, Metasploit, BypassUAC, OxidDumper, chimera_pe, Custom, Meterpreter, TinyMet, Pinjectra, ruler, Mapping-Injection, CookieMonster, SeatBelt, Rubeus, Lilith, myLittleRansomware, AsyncRAT-C#, PowerShdll, OffensiveCSharp, AmsiScanBufferBypass, TitanHide, SpoolSample, UnmanagedPowerShell, Watson, SCShell, Koadic, FruityC2, PoshC2, veil, Posh-SecMod, Empire, pupy, Trochilus, DKMC, PowerSploit, LaZagne, Apfell, FactionC2, SafetyKatz, BloodHound, reGeorg, iBombShell, FudgeC2, Callidus, SharpHound, KeeThief, jexboss, chrome-passwords, CHAOS, DeathStar, CrackMapExec, WinPwnage, Invoke-TheHash