2024 DUBLIN

2 - 4 October, 2024 / Dublin, Ireland

LEVERAGING AI TO ENHANCE THE CAPABILITIES OF SHAREM SHELLCODE ANALYSIS FRAMEWORK

Bramwell Brizendine

University of Alabama in Huntsville, USA

bramwell.brizendine@gmail.com

ABSTRACT

This paper explores the integration of AI to enhance the SHAREM shellcode analysis framework, a cutting-edge tool initially released at VB2022. SHAREM's unique capabilities include advanced deobfuscation, complete code coverage, and the ability to generate fully accurate disassembly of complex, encoded shellcode samples. The framework's evolution now incorporates AI-driven analysis to provide deeper insights into shellcode behaviour, offering automated interpretations and mapping malicious functionality to MITRE ATT&CK techniques. By utilizing AI models like GPT-4, SHAREM minimizes the need for expert analysis, providing highly detailed and comprehensive low-level analysis as well as high-level interpretations, both of which can help with malware analysis tasks.

INTRODUCTION

SHAREM was initially released at VB2022, as a game-changing shellcode analysis framework with some unprecedented capabilities [1]. Since then the tool has evolved greatly. With the ability to resolve tens of thousands of WinAPIs from among 60+ DLLs, in addition to virtually all user-mode Windows syscalls, SHAREM is able to tackle virtually any security-relevant function that might be encountered in shellcode. As SHAREM is an analysis tool, its goal is to discover as much functionality as possible, as often some network resources may be down, and thus it is important to simulate success as much as possible, so that we can discover functionality. Whilst that may not always be achievable, SHAREM pioneers a new approach we call 'complete code coverage', where all CPU and register data is saved, including snapshots of memory, so if a shellcode ends without parts of the code being visited, it can resume execution at that location, with CPU context restored, and it can then traverse the portion of code that was not previously visited. In so doing, it can discover any and all functionality that was otherwise missed. This process will repeat indefinitely, until all code has been traversed. These unique features will be reviewed briefly in the pages that follow.

Since SHAREM's release, the capabilities of AI have grown significantly, making it more accessible than ever before. AI has been leveraged to enhance the output of the framework. SHAREM is capable of producing an extraordinary amount of data on a given shellcode sample, but the fact remains that these are just points of data, open to interpretation. They may require a skilled set of eyes to understand what is going on. With the latest evolution of SHAREM, we try to go beyond that and remove the need for a highly skilled analyst. By leveraging AI, we can raise several important questions and provide answers, ultimately forming an insightful interpretation and analysis of a given shellcode sample. Whilst shellcode is used routinely in exploitation, we also can correctly regard it as a malware sample; after all, it is typically invoking some form of malicious functionality, directly or indirectly. Beyond knowing what WinAPIs and their associated arguments may be present, the AI can look at the big picture and determine specific techniques and sub-techniques that map to the MITRE ATT&CK framework. This correlation can offer much greater insights in terms of understanding what is happening. Analysis of the shellcode sample may suggest two or even three techniques. Not only that, but the tool provides an explanation, citing specific data from the shellcode sample. Using AI, SHAREM can produce a highly detailed report on what is being done in the shellcode, from the standpoint of MITRE ATT&CK techniques. At the same time the AI can look closely at the shellcode from a low-level perspective, offering interpretations as to what may be going on in terms of just the mechanics of the shellcode, without necessarily concern for the specific techniques that are being used.

Additional enhancements to SHAREM include its vastly improved complete code coverage, which can take into consideration even unusual circumstances. As we recognize that sometimes the analyst knows best, SHAREM provides opportunities for a more fine-grained approach to implementing the complete code coverage, as some more unusual control flow structures may warrant closer attention.

OVERVIEW

The following provides a brief overview of the previous capabilities of the SHAREM shellcode analysis framework.

Deobfuscation capabilities

SHAREM is able to take a shellcode with a single, simple encoding operation and decode it in seconds, using brute-force deobfuscation, if one of several simple encoding operations, such as xor, add, sub, etc., are present. It is able to do this with two or even three such simple encoding operations, even supporting distributed computing. However, a much more efficient way of deobfuscating the shellcode is via emulation, and this is the preferred method. SHAREM uses emulation to recover the true, hidden functionality, even with complex encoding algorithms. Additionally, SHAREM is unique in that it presents the deobfuscated form of the code in the disassembler by default, rather than the unintelligible, encoded form. This is true even if the shellcode re-encodes itself (which would happen only very rarely), in which case it is not just a simple matter of capturing the deobfuscated form, which potentially may never exist in memory – SHAREM would still be able to capture it by taking many snapshots at different points, based on different criteria, and then merging them together.

Enhanced disassembly

Most shellcode in disassembly has portions that are highly inaccurate, as code that may be data, such as strings or checksums, intermixed with the code, is misinterpreted as highly bizarre instructions. SHAREM has its own disassembler,

with its own analysis phases, which can produce markedly superior disassembly. However, it can also integrate emulation data with the disassembly it generates, and this can result in virtually flawless disassembly. As SHAREM performs what we call complete code coverage – meaning we ensure all parts of the code are emulated – all the code is accounted for.

Identification of WinAPIs and Windows syscalls

Shellcode does not utilize the Import Address Table (IAT) in the way a PE file does, instead preferring to abuse properties of Windows internals, such as PEB walking, etc. This means that disassemblers other than SHAREM cannot identify WinAPIs, let alone the more esoteric Windows syscalls. SHAREM is able to use its emulation data to allow for these to be clearly and cleanly identified, in vivid colours (as shown in Figure 1).

```
0x8f xor eax, eax                          31 c0                  1.
0x91 push eax                              50                     P
0x92 call dword ptr [edx + 0x159]          ff 92 59 01 00 00      ..Y...
    ;call to URLDownloadToFileA
      (0x0, http://127.0.0.1:9999/evil.hta,
       C:\Users\Public\evil.hta, 0x0, 0x0)
0x98 pop edx                               5a                     Z
0x99 pop ebp                               5d                     ]
0x9a push ebp                              55                     U
0x9b push edx                              52                     R
0x9c lea esi, [edx + 0x195]                8d b2 95 01 00 00      ......
0xa2 xor eax, eax                          31 c0                  1.
0xa4 push eax                              50                     P
0xa5 push esi                              56                     V
0xa6 call dword ptr [edx + 0x14d]          ff 92 4d 01 00 00      ..M...
    ;call to WinExec
      (mshta file://C:\Users\Public\evil.hta, SW_HIDE)
```

Figure 1: Even though this shellcode was fully encoded, we are able to see accurate disassembly with WinAPIs clearly labelled.

Simulating success

SHAREM must simulate success for all WinAPIs, as this can lead to it discovering more functionality. Otherwise, if a function were to fail, it might prematurely terminate, leaving some functionality undetected. Thus, SHAREM optimizes this to a great extent. It is not actually performing those functions, but it simulates success in memory, allowing the shellcode to find proper values. If an actual file is out there and available, it can be downloaded into memory and placed in the simulated file system.

Complete code coverage

This is one of the most groundbreaking features of SHAREM – the first of its kind – as SHAREM is able to utilize custom functionality to take snapshots of memory and CPU/register state at every possible branching location. If a shellcode ends without parts of the code being visited, SHAREM will restart at an unvisited location, with its previous memory and CPU state restored. This process ensures that all executable code, including previously unreachable sections, is executed by restarting from unvisited locations with the restored memory and CPU state. This increases the likelihood of triggering any potential behaviour. Though, that may not always necessarily be the case. In practice, this works very well. With the latest revision since VB2022, this functionality has been rewritten to be more efficient, and to take into account input from a VB2022 attendee who wondered how it could handle jump cases. SHAREM now provides more fine-grained capabilities for users to provide input into how it may respond at certain locations.

Timeless debugging

SHAREM also provides timeless debugging capabilities, by capturing every single instruction performed, and the register state before and after. A new feature allows users additionally to capture a snapshot of the stack before and after each instruction, or about 0x1000 bytes of it, although this does significantly slow down the emulation, so it is not recommended unless truly needed.

UTILIZING CHATGPT TO ADVANCE SHAREM'S CAPABILITIES

In trying to integrate AI with SHAREM, the question was whether or not ChatGPT could be utilized for this purpose. With some experiments, it was immediately clear that this could be done with careful prompt engineering, using a paid API key.

The two models tested have been gpt-3.5-turbo and the new gpt-4o. While both work, gpt-4o tends to give significantly better results. ChatGPT is already familiar with the MITRE ATT&CK framework, and gpt-4o seems to have an especially astute understanding. SHAREM is asking quite a bit of the AI, to perform tasks and evaluation in a certain prescribed way, and gpt-4o tends to be more effective. With some of the questions, the responses may be less detailed or insightful with gpt-3.5-turbo. Cost can also be an issue, as the usage of gpt-4o is approximately 1 cent per shellcode sample analysed, whereas gpt-3.5-turbo is much less. For an individual analyst who uses SHAREM only from time to time, this may not make a difference, but if it were to be deployed in bulk, then this cost consideration could be relevant. In general, the results obtained by gpt-3.5-turbo are not as good, but would be sufficient for gaining additional insights into a sample or identifying MITRE ATT&CK techniques. Gpt-3.5-turbo frequently does not identify as many MITRE ATT&CK techniques as the superior gpt-4o; the comments given by gpt-3.5-turbo tend to be less detailed and specific, but similar to those of gpt-4o. It is recommended that gpt-3.5-turbo only be used when dealing with bulk quantities of samples (hundreds or more), and where cost savings are important. As SHAREM can be deployed headless, it could be possible to deploy it routinely on all suspected samples of shellcode. Of course, there is no need to analyse shellcode if no APIs or Windows syscalls are discovered; some suspected, possible shellcode samples may be found not to be actual shellcode.

Other considerations with AI

SHAREM does make an API call to OpenAI; the discovered WinAPIs and syscalls, artifacts, and disassembly are shared – but not the actual binary sample. It is also important to note that OpenAI can be used in a closed environment, with additional setup, though this has not been tested with SHAREM.

LEVERAGING AI TO ANALYSE AND SEEK MITRE ATT&CK TECHNIQUES

SHAREM can utilize AI to analyse a shellcode and discover different techniques and sub-techniques from the MITRE ATT&CK framework. First, it can identify each technique, and then it provides a detailed description of why it considers a particular technique to be present. For instance, in one sample, the following were identified:

- T1105 - Ingress Tool Transfer
- T1059.003 - Command and Scripting Interpreter: Windows Command Shell
- T1219 - Remote Access Software
- T1140 - Deobfuscate/Decode Files or Information

Simply having the names of the identified techniques is insufficient; we want to understand why the AI identified these, and the tool provides the detailed logic behind such. For instance, with T1105, the description of this technique as it pertains specifically to that shellcode is as follows:

> ‘The presence of the URLDownloadToFileA function indicates an attempt to download a file from a URL, which aligns with the T1105 (Ingress Tool Transfer) technique. This is suggested by the URLDownloadToFileA function and associated URL.’

Additionally, SHAREM leverages AI to identify specific instances of the disassembly that pertain to a particular technique, such as ‘0x1200005f-0x12000076 (WSAStartup) 0x1200008a-0x120000b5 (WSASocketA) 0x120000b5-0x120000c5 (connect, recv, shutdown, WSACleanup)’, which are said to be ‘functions related to socket operations’, which ‘indicate the use of remote access software, aligning with T1219 technique’. With nearly all points of information, the specific locations in the disassembly are identified. With another shellcode sample, the T1059.003 sub-technique is discovered:

> ‘The CreateProcessA function executes “cmd.exe /c test.bat”, indicating the use of the Windows Command Shell to execute commands, which aligns with the T1059.003 technique.’

In another shellcode, several techniques are enumerated, and one of the techniques identified is T1070.001 - Indicator Removal on Host: Clear Windows Event Logs:

> ‘The command “netsh advfirewall set allprofiles state off” potentially clears firewall logs and makes it difficult to track network activity.’

In a different shellcode, the usage of GetComputerNameA is used to identify the T1016 - System Network Configuration Discovery technique:

> ‘The GetComputerNameA function retrieves the name of the current computer. This activity is consistent with adversaries attempting to gather information about the network configuration to understand their environment.’

Under the WinAPI subsection, additional details from the AI analysis offer insights into what is going on with GetComputerNameA:

> ‘The GetComputerNameA function is used to retrieve the computer name, “Desktop-SHAREM”, which can be helpful for network configuration discovery.’

In this case, Desktop-SHAREM happens to be the default NetBIOS name for the local computer, not anything discovered by SHAREM; the shellcode is made to believe that is the name.

In yet another shellcode sample, several MITRE ATT&CK techniques are identified. While some analysts undoubtedly could recognize the malicious potential with the WinAPI GetClipboardData, the identified technique, T1056.001 - Input Capture, makes its danger abundantly clear:

> 'The GetClipboardData function is used to capture clipboard data, which is consistent with the technique for capturing sensitive data that may be copied and pasted by the user.'

In the WinAPI section, which explores how WinAPIs are used with the MITRE ATT&CK techniques, more information is available from a security-relevant standpoint:

> 'Captures the current clipboard content, which might include sensitive information like passwords or keys.'

While likely readily apparent to analysts, this can provide quick and easy insights for less technical people who need to understand the significance of a potentially unfamiliar WinAPI. It is also possible they may have an understanding of its basic capabilities, but not from a security-relevant standpoint. As the shellcode sample abuses multiple registry functions, the T1112 - Modify Registry technique is identified:

> 'The RegSetValueExA function call to modify registry values is indicative of this technique.'

As SHAREM may identify multiple MITRE ATT&CK techniques, it can be beneficial to look at the big picture and figure out which one is most critical. As such, SHAREM leverages AI to identify the technique and provides a brief explanation as to why it is most critical. As there could be multiple techniques, figuring out which is the most severe or concerning could be useful. The following example explains why T1219 – Remote Access Software is the most critical of the techniques present in a given sample:

> 'This technique is most critical due to its implications for unauthorized remote access and command and control capabilities, posing significant risk for data exfiltration and system manipulation.'

Additionally, SHAREM provides a brief summary of all the techniques, providing an overview of all the techniques being used, as seen in the following:

> 'The analysed shellcode utilizes several techniques, including the creation of a command shell via CreateProcessA aligned with T1059.003, and setting up remote access capabilities via socket operations indicative of the T1219 technique. Furthermore, memory allocation and decoding routines are indicative of T1140. These techniques collectively demonstrate capabilities for remote access, command execution, and data handling.'

With another shellcode sample, we have the following:

> 'The shellcode employs multiple techniques to achieve its objectives. It uses the T1197 technique by invoking URLDownloadToFileA to download a file from a remote server. Post-execution, it employs the T1070.004 technique using DeleteFileA to delete traces of its activities. The CreateProcessA function is used to run commands through the Windows Command Shell, corresponding to the T1059.003 technique. These actions together indicate a complex chain of activities designed to download, execute, and clean up after execution.'

Whilst good writing would usually dictate avoiding so much quoting, in this case we are hoping to highlight the quality of output produced by SHAREM, as it accurately presents a coherent and complete narrative, detailing specific MITRE ATT&CK techniques used to achieve malicious functionality. The totality of what is being done is clearly explained with vivid, specific examples from the sample itself.

SHAREM is able to identify more than 25,000 WinAPIs used in shellcode, alongside the respective parameters. By leveraging AI, it additionally provides a definition of each WinAPI, and it provides insightful comments on the usage of each one, replete with specific details from the shellcode sample, and with a focus on how they might relate to MITRE ATT&CK techniques or other malicious functionality.

In one shellcode sample, initial activity starts with UrlDownloadToFileA, and the AI provide a usage description for that WinAPI, explaining how that WinAPI is being used:

> 'Downloads a file from "http://167.99.229.113/default.css" and saves it as "test.bat", indicating a potential download of a malicious script.'

Instead of having to deal only with parameters, we get a plain-language explanation of what is going on. Next, for CreateProcessA, the usage description is as follows:

> 'Executes "cmd.exe /c test.bat", suggesting the execution of downloaded or predefined commands.'

While SHAREM previously identified these artifacts and parameters, the AI allows it to go one step further and to offer some interpretation and connect the dots, so to speak; it explicitly tells us that CreateProcessA will likely result in executing commands contained in test.bat, so we do not need to draw those conclusions ourselves.

Next, for DeleteFileA, we have the following:

> 'Deletes "test.bat" after execution, aligning with the T1070.004 technique.'

The significance of what is being done is clearly explained to us, tying it to a known technique. The WinAPIs section of the AI report can be beneficial as it offers an analysis of how a WinAPI is being used as it pertains to malicious activities, offering insightful interpretations, with its usage often being correlated with known MITRE ATT&CK techniques. More experienced analysts can read the AI results and then manually inspect parameter values or examine the disassembly to see if anything was missed. That is, much of the work will already have been completed.

In another sample, the usage description for Sleep provides insights as to why the malware author may be using Sleep in the first place:

> 'Introduces a delay in the execution flow to possibly evade detection.'

SHAREM is able to discover and extract numerous artifacts from the shellcode's memory and from arguments used as parameters. Previously, SHAREM provided a list of artifacts organized into sub-categories. With AI, now all the artifacts can be analysed to create a coherent narrative that forms the Artifacts Summary, explaining the relevance of each artifact in the shellcode:

> 'The artifacts "urlmon.dll" and "test.bat" are significant in this sample. The "urlmon.dll" is required for downloading files using the URLDownloadToFileA function. The "test.bat" file is both downloaded and executed, containing commands to be run on the host system, and then deleted to remove evidence of the activity.'

This can be useful for quickly identifying artifacts that may have a security-relevant importance. The AI is able to take many artifacts and create a coherent narrative, explaining their relevance in the shellcode, as shown in the following:

> 'The main artifacts in the sample are the manipulation of the Windows Registry at HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run and the reference to calc.exe. These artifacts signify an attempt to achieve persistence by executing the calc.exe file upon system startup or user login, which could be a placeholder or a benign process used for testing purposes.'

An executive summary is prepared by SHAREM, by analysing the entirety of the shellcode, focusing on the big picture, without relying upon the lens of the MITRE ATT&CK framework. Instead, it offers a comprehensive overview of the malware's functionality, as seen here:

> 'The shellcode initializes network communication through Ws2_32.dll functions, establishing a remote connection and enabling data reception and transmission. It dynamically allocates memory, loads necessary libraries, and resolves function addresses at runtime. The shellcode also attempts to create a new process using cmd to execute command-line instructions. These behaviours collectively point to functionality aimed at remote access and command execution.'

Another executive summary provides a detailed assessment of malicious actions, while offering speculation as to what potentially could be done:

> 'The analysed shellcode appears to perform several malicious operations, including downloading a file from a remote server, executing it, and subsequently deleting it to cover tracks. The shellcode uses various WinAPIs such as URLDownloadToFileA, CreateProcessA, and DeleteFileA to accomplish these tasks. Additionally, the use of HTTP for communication indicates potential external interactions or C2 communications. The shellcode demonstrates techniques to evade detection and maintain persistence within the compromised environment. The most critical aspect of this shellcode is the use of URLDownloadToFileA to fetch additional malicious payloads, which could lead to further exploitation or lateral movement within the target network.'

SHAREM's leveraging of AI is not a replacement for the human analyst, as clearly the tool cannot fetch and analyse the payload. While SHAREM can download binaries if they are still live, it offers no further analysis other than hashing the payload.

As a cutting-edge analysis tool, SHAREM can identify both WinAPIs and Windows syscalls. The AI will also identify Windows syscalls, in the same fashion it does WinAPIs, labelling them correctly as syscalls, rather than WinAPIs. As Windows syscalls are very rarely used in shellcode, their significance is also explained. In a sample shellcode with no WinAPIs, but comprised entirely of Windows syscalls, the correct malicious functionality is identified with the appropriate technique, T1053 - Scheduled Task/Job:

> 'The NtCreateKey and NtSetValueKey functions are used to manipulate the Windows Registry, specifically to add an entry to the Run key, which ensures that the executable (calc.exe) is automatically executed upon system startup or user login. This behaviour is indicative of persistence mechanisms.'

If what was being done with these syscalls was not immediately clear, the identification of T1053 with specific details gives the needed illumination. The executive summary identifies the significance of using Windows syscalls:

> 'This shellcode primarily focuses on establishing persistence by modifying the Windows registry to ensure that an executable ("C:\Windows\System32\calc.exe") runs at every system startup. By utilizing native API calls such as NtCreateKey and NtSetValueKey, it creates and sets the necessary registry keys directly, bypassing standard Windows

> API calls and potentially evading detection by security software. Additionally, the shellcode cleanly closes handles and terminates the process, demonstrating a careful and stealthy approach to ensure no unnecessary lingering processes. The use of syscalls rather than WinAPIs indicates a higher level of sophistication, suggesting the author has a deeper understanding of Windows internals and is leveraging methods to avoid detection. The persistence mechanism through the registry is particularly concerning as it ensures the code runs on every system reboot, requiring manual intervention to remove.'

Using AI to help better understand what is happening with a shellcode sample offers tangible benefits. We are able not only to focus on low-level details, but also understand the totality of the damage they can inflict upon a victim. We don't just have the name of a technique identified, but we also have a clear recounting of why that determination was made, alongside supporting evidence. As the appearance of such are identified in the disassembly, the analyst can easily explore further when needed.

LEVERAGING AI TO ANALYSE SHELLCODE DISASSEMBLY

Analysis of shellcode can be done from a couple different perspectives. First, a tool could look at the different MITRE ATT&CK techniques and sub-techniques present, as explored above, and it could be used to look at how different WinAPIs are used to achieve malicious functionality. That is, a high-level view of the shellcode could be provided, without looking at the low-level mechanics of how the shellcode works.

SHAREM is the only shellcode analysis tool that also analyses the disassembly, providing fully accurate disassembly alongside identifying and annotating WinAPIs or Windows syscalls. Additionally, comments regarding the mechanics of how the shellcode works – above and beyond just malicious functionality – can be generated as well, such as identifying specific elements of the PEB walking process. For those who want a deeper, low-level understanding of the shellcode, we leverage AI further to analyse the sample and highlight the different shellcode techniques used, to enhance our understanding of the disassembly. This section of the tool, called 'Comprehensive Analysis of Disassembly and Data', is the subject of this part of the paper.

The Comprehensive Analysis of Disassembly and Data section first provides an 'Overview', which discusses shellcode functionality as well as mechanics of most shellcode samples, such as dynamic API resolution. Unlike with the previous section, the focus here is not on trying to tie the shellcode behaviour to MITRE ATT&CK techniques, but rather to give a more practical, low-level exploration of what is happening:

> 'The disassembly shows shellcode that prepares memory space, initializes network connections, dynamically resolves necessary functions, and executes commands through a new process. Notably, it includes typical shellcode patterns such as dynamic API resolution, memory allocation, and function calls for network communication.'

A section entitled 'Detailed Analysis' looks into specific components of the shellcode, providing valuable insights for those seeking to understand low-level details of how the shellcode works. The subsections include 'GetPC and Jump to Code Execution', 'Initial Setup and Calls', 'API Calls Setup', 'Function Resolution Using PEB Walking', 'API Pointers and Memory Addresses', 'Hashing Routine', 'Strings and Data', 'Function Names', 'Purpose of Recognized DWORDs (Checksum Values; Placeholders for API Pointers)' and 'Conclusion'. For each of these subsections, wherever possible, SHAREM identifies specific appearances of each shellcode feature in the disassembly.

The 'GetPC and Jump to Code Execution' subsection tries to identify techniques for capturing the program counter (PC), as this is often a requirement for shellcode:

> 'The shellcode starts with a call to itself to get the current program counter (PC) and adjusts the stack pointer.'

With another example, we have the following:

> 'The shellcode establishes its base address using call and pop instructions.'

As shellcode is position independent, it is necessary to capture a point of reference to memory for all but the most simplistic shellcode samples. While SHAREM already identified a GetPC gadget, this can provide additional commentary for more advanced examples.

The 'Initial Setup and Calls' subsection serves to identify some of the initial setup sequences, as can be seen in the following: '0x0-0xc: Shellcode entry and stack setup' and '0x12-0x18: Initializing variables and setting up for further calls'. Thus, if someone is not what sure is going on in that specific area of shellcode, its purpose can be identified.

The 'Function Resolution Using PEB Walking' subsection provides a straightforward explanation of parts of the code that deal with PEB walking. SHAREM already identified several specific features of PEB walking, but this extends it further, identifying ranges of code that help resolve runtime addresses for functions. The specific ranges of code include the following:

> '0x26e-0x2a3: Resolves the necessary functions by walking the Process Environment Block (PEB).'

Additionally, the AI is able to identify the cryptographic function used to help resolve the runtime addresses of WinAPIs – a feature which SHAREM previously did not identify:

> 'This likely includes a hashing routine seen in the range 0x291-0x296 (rol and xor operations).'

While the purpose of this may be readily apparent to experienced analysts, for those new to malware analysis, the purpose of this section of the disassembly could be cryptic. Hence, SHAREM's usage of AI can provide some highly beneficial insights. If Windows syscalls are used instead of WinAPIs, then this subsection will not contain information.

SHAREM also separately identifies the hashing routine, if present, that may be used to help with the dynamic resolution of WinAPIs. Often with shellcode, the string name of each WinAPI is hashed, and that resulting checksum is compared with a precomputed checksum for a desired WinAPI. This enables malware authors to utilize WinAPIs without having to use their strings, adding to stealth and allowing the target WinAPI to be resolved in a more covert fashion. This is often done with custom hashing algorithms, though many are reused from previous malware. The AI is able to identify and analyse the specific features of the hashing routine, pinpointing the specific locations it may appear in the disassembly, as seen in the following:

> 'The hashing routine appears in the range 0x283-0x298, using a combination of rol and xor to compute hashes.'

In another shellcode example, this description comes under 'Hashing Routine':

> 'It calculates a hash of a string for function until it finds the match of a pre-calculated hash.'

Immediately below this, an example of the key elements of the hashing algorithm is provided:

> '0xe8 rol edx, 7; 0xeb xor dl, al.'

When analysing code, a function may be renamed to help better indicate its functionality. Often an analyst may do this to aid their understanding. Shellcode can use multiple functions internally. SHAREM is able to recognize what a function is and what is done within it. This enables it to suggest descriptive function names that may indicate functionality, e.g.:

- label_0x5: Shellcode Entry Point (0x0-0x0)
- label_0x191: LoadLibraryA Call (0x191-0x1a5)
- label_0x26e: PEB Walking (0x26e-0x2a3)
- label_0x2a4: Function Resolution (0x2a4-0x308)

While SHAREM does not presently rename functions internally with the above suggestions, a user could use this to rename it in other tools, such as *Ghidra*.

Fully accurate disassembly is based on being able to properly distinguish between data and code or instructions. If this can be achieved, then the disassembly should be nearly flawless, but what is to be made of the data? How is that to be represented? SHAREM has its own custom functions to identify strings, which are correctly represented as such, rather than being misinterpreted as instructions. But what of more cryptic forms of data? SHAREM's emulation tracks memory reads and writes to specific locations, and it is able to use this to help identify data, even if code is self-modifying. Thus, SHAREM can identify some code as simply DWORDs – data whose purpose may not be known. However, now with AI, it can identify some of these DWORDs as being checksums used for WinAPI dynamic resolution. For instance, a checksum might be used to identify WSAStartup when iterating through an array of WinAPI names. With AI, SHAREM will identify certain ranges of addresses as checksums, clarifying their purpose: 'Used for hashing module names to facilitate dynamic function resolution.'

Another shellcode results in the following description: 'These appear to be checksum or hashed values used in the hashing routine to resolve API addresses,' while identifying the checksums in question. Elsewhere above, the shellcode identifies the key elements of the hashing algorithm, the results of which are later compared with precomputed hashes. Additionally, shellcode often uses arrays of DWORDs, some of which are placeholders that will later be replaced with the runtime address of a function. SHAREM was already able to identify these without the use of AI, indicating, for instance, that a certain location might be an API pointer for URLDownloadToFileA. The AI repeats some of this, with additional, minimal comments.

The 'Conclusion' subsection of the 'Comprehensive Analysis of Disassembly and Data' section does not deal with MITRE ATT&CK techniques or a high-level view of malicious functionality in a vacuum, but instead it gives concluding remarks on all the functionality of the shellcode, including routine mechanics of API resolution, as in the following example:

> 'The shellcode performs initial setup routines to establish a controlled execution environment, dynamically resolves networking-related API functions, and attempts to establish network communication for potential remote command execution. It terminates cleanly after executing the intended operations.'

Another shellcode offers the following:

> 'The shellcode effectively downloads a malicious file from a remote server and executes it, then deletes the file to avoid detection. It uses several Windows APIs to accomplish these tasks, making use of memory allocation and API function resolution techniques. The shellcode demonstrates common malicious behaviours including file downloading (T1105), process creation (T1059.003), and file deletion (T1070.004). The absence of an extensive hashing routine suggests straightforward API resolution and execution.'

GHIDRA PLUGIN

In 2023, Max Kersten of *Trellix* created a plugin for SHAREM, allowing its results to be integrated into *Ghidra*. *Ghidra* and *IDA Pro* are unable to identify WinAPIs being called in the disassembly. More often than not, it would just be an ambiguous indirect call to some register. One of SHAREM's claims to fame is that it can identify each function call – be it WinAPI or Windows syscall – along with parameters, making interpretation of disassembly much easier. This is the case even with encoded shellcode, which can be transformed to reveal its inner secrets. While SHAREM has its own custom disassembler, it is not fully featured. As such, the idea was to extend SHAREM's capabilities to *Ghidra*, allowing SHAREM to be run headless, and for its results to be integrated into *Ghidra*. For additional details, please refer to documentation on *GitHub* and check out the Sharem.java script at [2]. Thus, for times when more intensive work is needed when working with a shellcode in disassembly, using it in *Ghidra* is a viable option.

FINAL REMARKS

SHAREM remains a game-changing addition to our arsenal when it comes to analysing shellcode. While it is not the only tool to provide analytical capabilities for shellcode, it far exceeds any other tool, with its many unprecedented features. Now, thanks to ChatGPT, we can leverage AI to extend SHAREM even further and produce intelligent and well-supported analysis of a given shellcode sample, based on the numerous features found by SHAREM.

ACKNOWLEDGEMENTS

SHAREM was supported by NSA Grant H98230-20-1-0326.

While this paper is focused on leveraging API to enhance SHAREM results, we also acknowledge the excellent work done by student collaborators: Austin Babcock, Jake Hince, Shelby VandenHoek, Sascha Walker, Evan Read, Dylan Park, Tarek Abdelmotaleb and Kade Brost. We also thank Max Kersten for his independent work with SHAREM in extending it to be available as a plugin on *Ghidra*.

REFERENCES

[1] Brizendine, B.; Hince, J.; Babcock, A.; Abdelmotaleb, T.; Walker, S.; VandenHoek, S. SHAREM: shellcode analysis framework with emulation, a disassembler, and timeless debugging. Virus Bulletin. September 2022. https://www.virusbulletin.com/conference/vb2022/abstracts/sharem-shellcode-analysis-framework-emulation-disassember-and-timeless-debugging/.

[2] Trellix Advanced Threat Research. GhidraScripts. https://github.com/advanced-threat-research/GhidraScripts/tree/main.