|  |  |
| --- | --- |
| Iterative Security Test 7 Report | Version: **#** |

# Project Name:

Test Completed on: Date

Team Members:

BLUF Statement: Project is on schedule, there are currently no issues blocking progress.

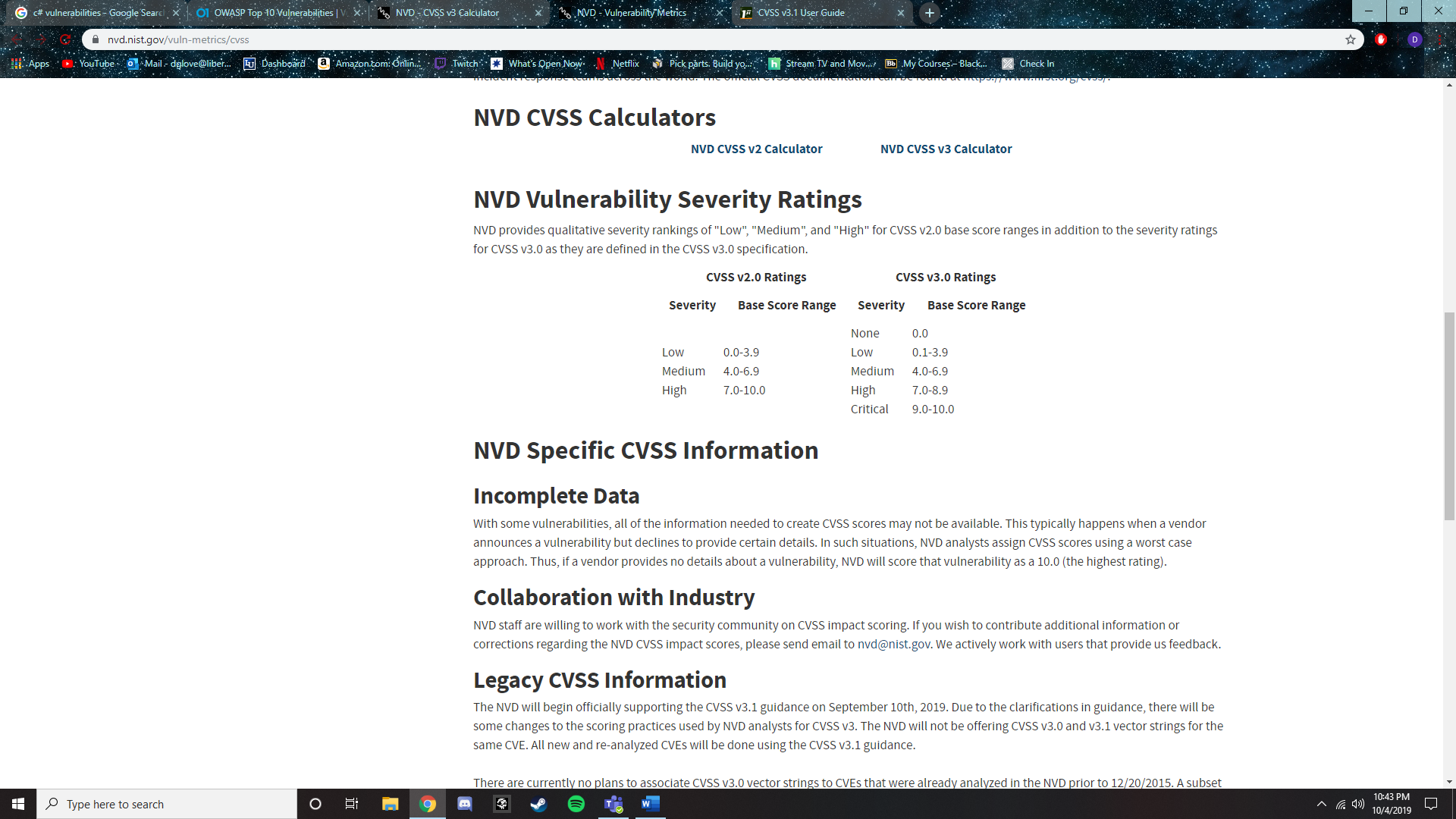
|  |  |  |
| --- | --- | --- |
| Vulnerability/Risk Level: |  |  |
| * Low | * High | * Untested |
| * Medium | * Critical |  |

**Executive Summary**

EXAMPLE: From the initial code that was given, multiple risks and requirements were determined. Based on the information that was given on the entire environment for Velocity, it was determined that there are two risks in the OWASP top ten risks that are apparent. Because the software within our scope is not open source or COTS (Consumer off the Shelf) software, it is highly unlikely that Common Vulnerabilities and Exposures (CVEs) exist for the software. Therefore, CVEs were not considered during this test.

The National Vulnerability Database’s common vulnerability scoring system is used for all test case’s risk levels. This system classifies risks and vulnerabilities on a scale from 1.0-10.0, with each level having a variety of characteristics. Because of redundancy, we have not included the initial ‘none’ risk level, but rather introduced a risk level of ‘untested’ that will help the client and team keep track of what has not been tested yet.

The Iterative Security Test 7 Report completed on November 17, 2019 consists of a further completion of the Manual Code Review Findings, Automated Code Review Findings, Penetration Testing Findings, Summary Graphs, the Action Items and Areas of Interest sections, API Development Process, Security Recommendations, and Acronym Definitions.



*Figure 1*. Vulnerability Metrics. Retrieved from <https://nvd.nist.gov/vuln-metrics/cvss>.

**OWASP Top 10 Findings**

EXAMPLE: The two OWASP top ten risks deemed relevant to the scope of the project are the Man in the Middle (MITM) and Denial of Service (DOS) or Distributed Denial of Service (DDOS) attacks. The MITM attack intercepts communications between systems by dividing the TCP connection into two connections between the client and attacker and the attacker and the server (Man-In-The-Middle Attack, 2015). The DOS attack restricts the use of a website, application, or server to a user by manipulating network packets. An attacker can use this method to flood service and the service can become unavailable to users (Denial of Service, 2015). Because these attacks are network-based, they cannot be tested with a manual source code review, therefore they were not tested during the baseline security test. These risks will be assessed during the iterative testing cycle, specifically during penetration testing.

For descriptions of security test cases and their associated risks and security requirements, see Appendices B and C.

|  |  |  |
| --- | --- | --- |
| Risk | Test Cases (See Appendix C – Security test Cases) | Status |
| **Man in the Middle Attack** | TC-PT-001: PenTest: Man in the Middle   * SR-AUTN-001 * SR-AUTN-002 * SR-AUTN-003 | Not Tested |
| **DoS/DDoS (Denial of Service and Distributed Denial of Service)** | TC-PT-003: PenTest: DOS/DDOS   * No related security requirements, but DOS/DDOS attacks are a risk that need to be assessed. | Not Tested |

**CVE’s**

After review of the different technologies being used in this product, we decided to research different CVE’s that may be applicable. We previously believed that since the product being tested is a proprietary product, that there would be no applicable CVE’s. We decided that we should consider the technologies being utilized by the product such as .NET core 3.0 which is being used by the API. After a lengthy search of CVE’s associated with .NET core 3.0, there were no results. All CVE’s associated with .NET core were in regard to version 2.2 and earlier.

As more technologies are discovered to be in use by the product, we will reevaluate associated CVE’s. Until then, we will stand with the assumption that because our project is not based on consumer off the shelf, open source, or otherwise publicly available software, no common vulnerabilities and exposures (CVE’s) exist for the proprietary software that makes up the project. CVE’s exist for the platforms that run the software, but the platform is not in the scope of our security testing. Therefore, CVE’s will not be addressed at this time

|  |  |  |
| --- | --- | --- |
| CVE | Test Cases | Status |
| N/A | N/A | N/A |

**Manual Code Review Findings**

The purpose of the Manual Source Code Review is to determine areas of interest for the iterative testing cycle, as well as document and communicate the initial security posture of the project’s software.

For descriptions of security test cases and their associated risks and security requirements, see Appendices B and C.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Cases & Associated Risks and Requirements | Discovered Vulnerabilities & Impact Summary | Assigned To: | Status |
| **[Charity Barker]**  TC-MSCR-001- API Authentication Bypass:   * SR-IDEN-001 * SR-AUTN-002 * SR-AUTN-003 * SR-CONF-001 | During the manual source code review, no evidence of the use of API authentication was found.  SR-IDEN-001: Unique Identifiers are used for each client.  SR-AUTN-002: Untested – No API authentication means that no credentials are stored.  SR-AUTN-003: Untested – No API authentication means that reauthentication is nonexistent.  SR-CONF-001: API calls are made using the .NET HttpGetAsync GET requests that do not protect (encrypt) data in transit. | Charity Barker | In Progress |
| **[Charity Barker]**  TC-MSCR-002 – Privilege Escalation:   * SR-IDEN-002 * SR-AUTR-001 * SR-IMMU-002 * SR-IMMU-003 * SR-IMMU-004 * SR-SURV-001 | Upon initial inspection, it is likely that the application can be abused to either gain initial access to a host machine and/or gain elevated privileges on a host machine. No proof of concept exploitation paths have been  SR-IDEN-002: No backdoors were discovered during the code review.  SR-AUTR-001: After packages are downloaded, their origin is not verified before their contents are used by the software.  SR-IMMU-002: The software is divided between a system agent and a user agent; however, no evidence of other access management was discovered.  SR-IMMU-003/4: Downloaded packages are allowed to run indiscriminately; the client application does not restrict the types of information that can be collected or the commands that can be run.  SR-SURV-001: C# is resistant to buffer overflows, unless the “unsafe” keyword is used. A full text search revealed zero uses of the “unsafe” keyword. |  | Not Resolved |
| **[Aaron Demers]**  TC-MSCR-003 – COM Hijacking:   * SR-AUTN-002 * SR-IMMU-002 * SR-IMMU-003 | COM is not used by the code within the scope of the project. If we end up working with the software engineering team on their package that interacts with applications, COM Hijacking will become an area of interest. |  | Resolved |
| **[Aaron Demers, Charity Barker]**  TC-MSCR-004 – Logging:   * SR-AUDT-001 * SR-AUDT-002 * SR-AUDT-003 * SR-AUDT-004 * SR-INTG-004 * SR-NREP-001 * SR-NREP-002 * SR-SURV-002 | SR-AUDT-001: Sufficient application logging is done to map a chain of events.  SR-AUDT-002: Specified security event logging is unknown at this time.  SR-AUDT-003: After 20MB of event storage, log overwriting is used. No log archival is implemented.  SR-AUDT-004: Application logs are accessible through the Windows Event Viewer application. The logs contain enough data fields for a system administrator to efficiently parse and sort data.  SR-INTG-004: Log integrity checks are not implemented; however, the application does check if logs are able to be written.  SR-NREP-001: The application does not log the reception of packages or data from the server. It does log the uninstallation of packages from the application.  SR-NREP-002: Application logs that are accessible through the Windows Event viewer are timestamped.  SR-SURV-002: Some logging malfunctions are logged (if the log does not exist) but not all possible/probable malfunctions are tracked.  **The issue will be marked as resolved when security recommendations are delivered to the customer, including modified source code.** | Charity Barker | In Progress |
| **[Heidi Waddell]**  TC-PT-008: Back Doors  SR-IDEN-002 | Simply stated, a backdoor is a type of software design flaw that allows an attacker to bypass appropriate authentication methods to gain access to the system. As displayed in Figure 1 included below, the ‘Win32Error’ function in the ‘RegistryKey.cs’ file in Velocity throws specific exceptions when an ‘errorCode’ integer has the values of 2, 5, and 6. All other values receive a generic Input/Output error. The ‘RegistryKey.cs’ file also contains a ‘Win32ErrorStatic’ function, displayed in Figure 2 below, which throws exceptions if the errorCode is any value other than 5 or if the string value is NULL. Based on these safeguards, registry modifications that would usually affect the application are checked within the functions of the application. Although some backdoors can be impossible to find from a manual source code review due to obfuscation, the source code provided to the team is of a very basic nature and therefore it is highly unlikely that obfuscated backdoors exist in the application. |  | Not Resolved |

Screen capture of code  
*Figure 2*. Win32Error Function*.*

Screen capture of code  
*Figure 3.* Win32ErrorStatic Function.

**Automated Code Review Findings**

After further investigation for installing an extension, Puma Scan, it was determined that Visual Studios Code was not compatible with any source code reviewer. Due to this, Visual Studios Community was installed for ease of compatibility with software security reviewers. The purpose of source code review is to identify code-level problems to prevent security vulnerabilities as early as possible. Static code analyzers find and list the security risks in the source code. The findings of the source code review will be defined and ranked in the bug priority matrix.

P[uma Scan](https://pumascan.com/) - Puma Scan is a .NET C# open source static source code analyzer that runs as an IDE plugin for Visual Studio. Puma Scan is a software security analyzer that provides real time and continuous source code analysis for C# applications as team members write and develop code. Puma Scan hunts for vulnerabilities identified in the OWASP Top 10, such as, SQL Injection, Deserialization Vulnerabilities, and Cross-Site Scripting (Puma Scan, n.d.).

Puma Scan is a background task that scans code as the user is writing their code in Visual Studios. Puma Scan follows all the processes of a third-party program all while working along Visual Studios. While writing code, Puma Scan will make suggestions to help the user write safer and more efficient code for commands by referring the user to a commonly practiced method for the same command. Puma Scan checks for Cross-site scripting (XSS) vulnerabilities (Puma Scan, n.d.).

For descriptions of security test cases and their associated risks and security requirements, see Appendices B and C.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Cases & Associated Risks and Requirements | Discovered Vulnerabilities & Impact Summary | Assigned To: | Status |
| **[Kaitlin Weathers]**  STC-ASCR-001 – API Authentication Bypass   * SR-IDEN-001 * SR-AUDT-002 | The .NET framework utilizes the PasswordDeriveBytes Class within the PasswordDeriveBytes.cs file to derive a key from a password using the PBKDF1 algorithm (PasswordDeriveBytes Class, n.d.). Proper authenticated is necessary for access. Tokens are used to validate a user’s request by containing the information of the username, timestamp, IP address, etc (Becker, 2015). The PBKDF1 algorithm is used to reduce vulnerabilities against brute force attacks (Avalanche Effect Analysis on PBKDF1 Algorithm, n.d.). The PBKDF1 algorithm in the code utilizes a key size of fewer than 16 characters. The Obsolete class within the PasswordDeriveBytes.cs file replaces PasswordDeriveBytes with Rfc2898DeriveBytes and is the preferred way to derive key material from a password and uses the *PBKDF2 algorithm (PasswordDeriveBytes Class, n.d.)*. Further investigation will continue to determine if a password is randomly generated and how secure the PBKDF1 algorithm operates. A Brute-force attack could be performed to attack authentication and discover hidden contents in an application. | Charity Barker | In Progress |
| **[Kaitlin Weathers, Daniel Love]**  STC-ASCR-002 – Malicious Binary Delivery   * SR-AUTR-006 * SR-INTG-003 | Examining the installer files (projectinstaller.cs) for the client, it was determined that binary can be injected through download of the installer. At this point in time, it was unable to be determined if the installer code had any code that could be taken advantage of, mainly because this type of attack is usually found outside of the code. Marked as not resolved now but might not end up being a problem with specifically the installation file. After further study, it was determined that we checksum needs to be incorporated. This will need to be discovered, but while the search is still in progress, monitoring of packet transfer can be used as an alternative. | Daniel Love | In Progress |
| **[Dylan Van Reenen]**  STC-ASCR-003 – DNS Hijacking   * SR-AUTN-001 * SR-AUTN-003 | The .NET framework deals with DNS resolution using a class titled System.Net which contains functions such as Resolve (resolves a DNS host name or IP address) and GetHostByName (gets the DNS information for the specified DNS host name). After review of the code, multiple source code files were found to contain the System.Net class. However, none of the source code use any of the functions dealing with DNS. As a result, we can conclude that DNS is not handled in the code. Will mark as resolved as of now. If evidence is found to contradict this, the test will be conducted again. | Dylan Van Reenen | Not Resolved |

**Automated Dynamic Testing Findings**

Automated dynamic testing involves checking the response of the system to the application being run. It observes the behavior of the software system, memory usage, CPU usage, and overall performance of the system1. The main goal of automated dynamic testing is to ensure that the finalized product is in a correct working state that does not overexert the machine to unstable levels. Automated dynamic testing can be conducted using unit testing, integration testing, and system testing. We have determined that this type of testing is not within the scope of the customer’s goals for the project. As a result, automated dynamic testing will be omitted from our testing strategy unless a reason is found to include it.

**Penetration Testing Findings**

The purpose of the penetration test is to simulate ways in which an adversary attacking a network running the software would interact with and attack the software. Due to the large number of test cases in the penetration test, it will be split across multiple iterative tests to allow each case to be fully and carefully tested.

For descriptions of security test cases and their associated risks and security requirements, see Appendices B and C.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Cases & Associated Risks and Requirements | Discovered Vulnerabilities & Impact Summary | Assigned To: | Status |
| **[Charity Barker]**  TC-PT-001: Man in the Middle   * SR-AUTN-001 * SR-AUTN-002 * SR-AUTN-003 | The testing and development network is not configured in a manner that allows network traffic from one machine to be collected and viewed by another. This makes is difficult to test Man in the Middle attack testing. The action item related to this test case for the upcoming week is to see if the testing environment can be reconfigured to allow collection of network traffic. |  | Not Tested |
| **[Dylan Van Reenen]**  TC-PT-002: DNS Hijacking | The .NET framework deals with DNS resolution using a class titled System.Net which contains functions such as Resolve (resolves a DNS host name or IP address) and GetHostByName (gets the DNS information for the specified DNS host name). After review of the code, multiple source code files were found to contain the System.Net class. However, none of the source code use any of the functions dealing with DNS. | Dylan Van Reenen | Not Resolved |
| **[Kaitlin Weathers]**  TC-PT-003: DOS/DDOS   * SR-ATEN-001 * SR-ATEN-005 | The .NET framework contains the CompressedStack class and is defined within the CompressedStack.cs file. The CompressedStack class represents the code access security information containing a Deny action (CompressedStack Class, n.d.). The CodeAccess Permission.Deny Method to prevent callers higher in the call stack from using the code (CodeAccess Permission.Deny Method, n.d.). | Kaitlin Weathers | In  Progress |
| **[Charity Barker]**  TC-PT-004: API Auth Bypass   * SR-IDEN-002 | After inspecting the source code, the team has decided to ask the client before testing the API. Because the API may be a production service, we’d rather not attack it without express permission and further information from the client.  Per the client meeting on 10/11/19, we will not be given access to the API to preform testing. API Authentication security recommendations will be documented and delivered to the client. This test case will be marked as resolved once the client receives the security recommendations. | Charity Barker | In Progress |
| **[Dylan Van Reenen]**  TC-PT-005: Malicious Binary Delivery   * SR-INTG-001 * SR-INTG-002 * SR-INTG-003 * SR-IMMU-001 * SR-SYSM-002 | After review of the source code, it has been determined that the best way to conduct malicious binary injection is to create a malicious package to be installed and run by the application. Another possibility is to modify an existing package before installation. However, this will be difficult without prior knowledge on how the packages are structured. According to the file ‘AgentUtilities.cs’ a package is downloaded using the function ‘DownloadPackageVersion’ from http://....and saved as a .zip file in | Dylan Van Reenen | Not Resolved |
| **[Dylan Van Reenen]**  TC-PT-006: Privilege Escalation   * SR-IMMU-002 * SR-IMMU-003 * SR-SURV-001 * SR-SYSM-002 * SR-AUTR-001 * SR-AUTR-002 | The System agent code takes advantage of the System.ServiceBase class. This class utilizes “ServiceProcessInstaller” which is responsible for installing the service onto the system. “ServiceProcessInstaller also specifies which account is going to be used to install the service on the system. In “ProjectInstaller.cs” there is a function “InitializeComponent” which specifies that the LocalSystem account will be used to install and start the process contained in the package fed into the agent. The LocalSystem account has full access to the resources on the machine ([source](https://docs.microsoft.com/en-us/windows/win32/services/localsystem-account)). If a malicious package is able to be fed into the agent, we could use the LocalSystem account to our advantage. |  | Not Resolved |
| **[Aaron Demers]**  TC-PT-007: COM Hijacking   * SR-AUTN-002 * SR-IMMU-002 * SR-IMMU-003 | COM is not used by the code within the scope of the project. If we end up working with the software engineering team on their package that interacts with applications, COM Hijacking will become an area of interest. |  | Resolved |
| **[Heidi Waddell]**  TC-PT-008: Back Doors   * SR-IDEN-002 | At this time, the risk of backdoors in the code has been assessed through a manual code review. Based on the safeguards built-in to Velocity’s source code, the risk of backdoors is at a low level, and therefore is not at a high priority for testing. However, penetration tests to ascertain the risk of backdoors in the code will be performed in the future. |  | Not Tested |

# **Summary Graphs**

*Figure 4*. Current Unresolved Vulnerabilities by CVSS Severity*.*

*Figure 5*. Discovered Vulnerabilities by Category*.*

*Figure 6*. Vulnerability Resolution Progress*.*

The Required Diamond Threat Model will be placed here.

*Diamond Threat Model For Application*  
**Action Items & Areas of Interest**

In the next phase of testing, the Cyber team will be addressing issues that were discovered during the manual source code review in this baseline security test. The vulnerabilities that were discovered will be investigated further and tested for their potential level of risk, and security recommendations will be made based on these findings.

* **TC-MSCR-001- API Authentication Bypass:**
  + Discover and investigate the local API. This will begin with the content discovered on the localhost 127.0.0.1.
  + Although the Cyber team does not have access to or control over the API, basic security tests can be performed on the authentication and authorization being implemented in it. From these discoveries, recommendations for securing these functions will be created.
* **TC-MSCR-002 – Privilege Escalation:**
  + Preform testing and analysis to better understand the usage and delivery method of the packages that get run by the agent.
  + Further investigate the differences between the system agent and the user agent. Questions to answer include:
    - Which ones handle specific networking functions? Is there segmentation implemented?
      * If so, how and why?
    - Which one runs with elevated permissions, and what does it interface with?
* **TC-MSCR-004 – Logging:**
  + Research and examine the implementation of a more robust system of logging, and investigate the creation of a patch to apply these changes. Currently, the system of logging is decently secure, but gaps remain and more security is feasible and within scope.
  + Currently documenting logging improvements to be discussed at the next client interaction.
* **TC-PT-001 – Man in the Middle:**
  + Modify the testing environment to allow collection of network traffic.
* **TC-PT-002- DNS Hijacking**
  + Continue to check if there is any possibility of gaining admin rights on the user system in order to redirect DNS traffic.
* **TC-PT-004 – API Authentication Bypass:**
  + Request explicit permission from the customer before accessing the API to determine the type or authentication required, from a pertest point of view.
  + Developing API Security Recommendations (and presentation) to deliver to the customer on 11/1/2019.
* **STC-ASCR-002 – Malicious Binary Delivery**
  + Determine whether the installer code can be injected into, or if a different mode of injection is used to gain access to a modified installer.
* **Other:**
  + Various places inside the packagehelper.cs file mention a ‘Test Mode’ (see Figure 6 and Figure 7 below). As the team is unsure what this value is and what it does inside the code, steps will be taken to discover its meaning and discover any possible vulnerabilities.

*Figure 7*. testMode Value*.*

*Figure 8*. testMode Declaration*.*

**API Development Progress**

At this point in the testing process, we are working towards the development of our own API to connect with the code provided to us by the team. So far, we have a working example of an API that displays weather information. In the next week, we plan to successfully configure the API to receive in the information sent by the System Agent and display it in an easy to understand format. We also plan to implement OAuth 2.0 authentication into the API to cover the hole in security dealing with authentication.

To create a basic API that will properly interact with the code supplied by, a new folder for a custom API will be created with new code that will be written by the team, and modifications to the existing code will be necessary. These modifications include changes to the following functions, which the software uses to connect to their own API. First, modifications to the AgentService.cs file will change the OnStart function, which contains the call to the API URL. In InteractiveControllers.cs, changes will be made to the ApiController to direct the processes to the new API. Other changes may also be necessary.

Screen capture of code  
*Figure 9.* OnStart Function located in AgenService.cs.

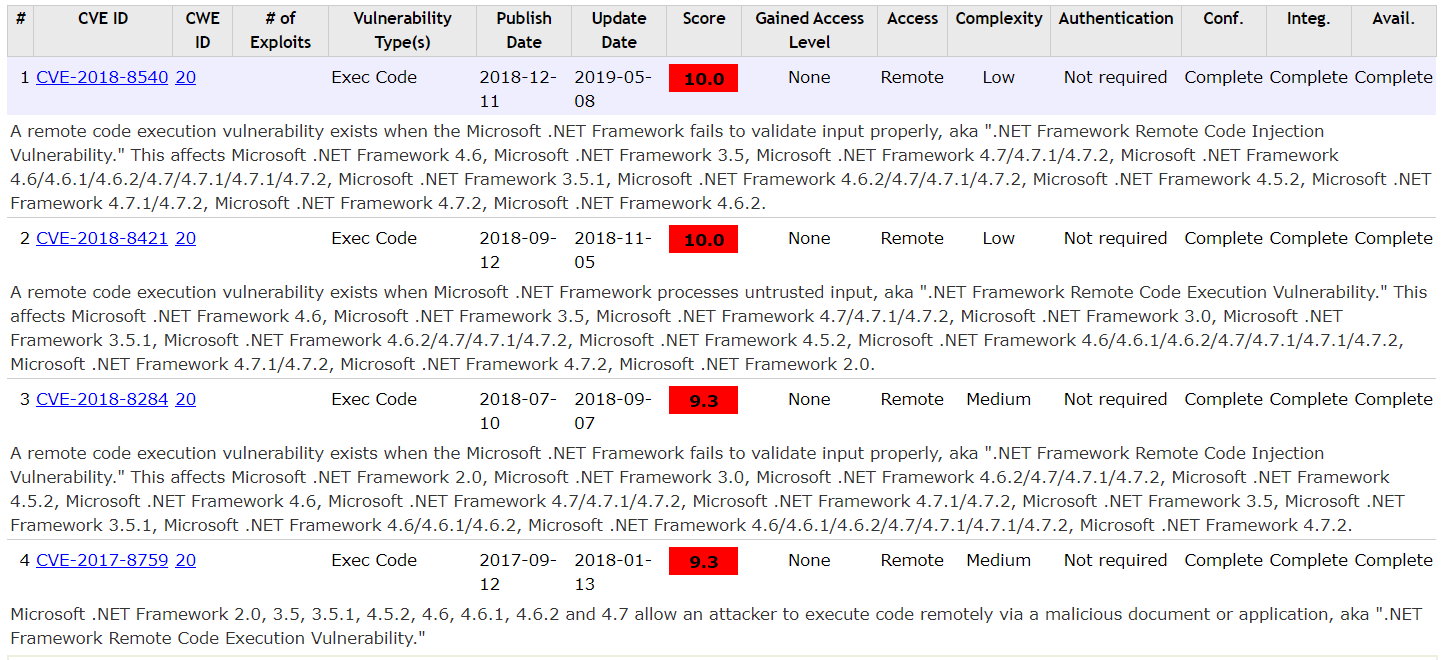
*Figure 10.* ApiController class in InteractiveControllers.cs.

**Security Recommendations**

**Security Vulnerabilities in .NET Framework 4.6.2**

Currently, the Velocity application API runs on the .NET Framework version 4.6.2/1. There are 25 currently identified common vulnerabilities and exposures (CVEs) in this version of the framework, 4 of which are critical vulnerabilities (scored above a 9). All four of these vulnerabilities are ‘Exec Code’ types, meaning that an attacker can remotely execute malicious code on the host system. Two of these vulnerabilities are particularly dangerous; not only do they have a CVSS score of 10.0, but they also have a low complexity rating, meaning that the attacker needs very little knowledge or skill to exploit it. The first of these is CVE-2018-8540, in which the .NET Framework does not validate input properly and opens the system up to a remote code execution vulnerability. The impact of this CVE is extreme; when exploited, all information is disclosed (all system files are revealed), all integrity is lost, and all affected resources can be made completely unavailable. The second, CVE-2018-8421, lies in the framework’s processing of untrusted input. If exploited, this vulnerability also results in complete loss of confidentially, integrity, and availability. The other two critical vulnerability, both ranked at a 9.3. CVE-2018-8284 first is a remote code injection vulnerability that occurs following faulty input validation, and CVE-2017-8759 allows a malicious document to be executed remotely. In addition to these four critical CVE’s, there are also two CVE’s ranked above a 7.0 score. One of these, CVE-2018-8202, allows for privilege escalation, while the other, CVE-2017-0160, is an additional remote code execution vulnerability.

The detailed summaries for the four critical vulnerabilities are as follows:

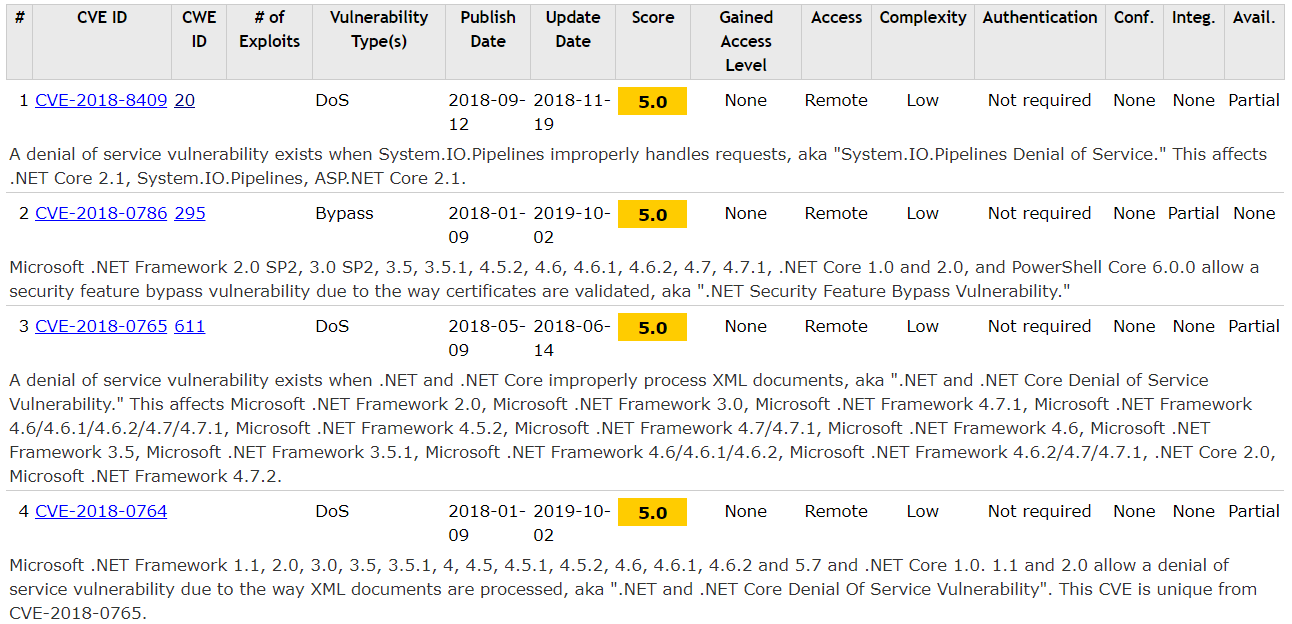
   
*Figure 11.* Critical CVE’s in the .NET Framework 4.6.2.

In order to remove these vulnerabilities, Microsoft recommends updating to the latest version of the .NET Framework, which contains various security patches to correct these vulnerabilities.

**.NET Core 2.2.7 Vulnerabilities**

While the company currently uses the .NET Framework, version 4.6.2, it is possible that a switch to .NET Core 2.2.7 will occur in the future. Therefore, the vulnerabilities existing in this API are also considered. Currently, there are only five vulnerabilities known to exist in this framework, with the top four scored at a 5.0. Three of these are Denial of Service type vulnerabilities. CVE-2018-8409 exists when the system pipeline handles requests improperly, resulting in a DoS vulnerability. CVE-2018-0765 is a Dos vulnerability existing in the improper processing of XML documents, and resides in multiple versions of .NET Core. CVE-2018-0764 is similar to this, as it is an XML document processing vulnerability. Finally, CVE-2018-0786 is a Cypass type vulnerability, which allows a bypass vulnerability based on the way certificates are validated.

The detailed summaries for these four medium vulnerabilities are as follows:

  
*Figure 12.* Medium CVE’s in .NET Core 2.2.7.

To address these vulnerabilities, Microsoft recommends implementing the September 2019 updates to .NET Core 2.2.7 and .NET Core SDK, which contain security and reliability fixes.

**Security Vulnerabilities in OAuth 2.0**

Starting December 17, 2019, OAuth 1.0 will no longer be able to be used and OAuth 2.0 will have to be used. There are vulnerabilities in OAuth 2.0 that could lead to an account takeover. The vulnerabilities stem from the implementation details

# Acronym Definitions

OWASP – Open Web Application Security Project

CVE – Common Vulnerabilities and Exposures

CVSS – Common Vulnerability Scoring System

MITM – Man in the Middle Attack

DOS/DDOS – Denial of Service / Distributed Denial of Service

MSCR – Manual Source Code Review

ASCR – Automated Source Code Review

PT – Penetration Test, Pentest

TC – Test Case

SR – Security Requirement

# References

Avalanche Effect Analysis on PBKDF1 Algorithm. (n.d.). Retrieved October 12, 2019, from GitHub: https://github.com/Aitordev/Avalanche\_PBDFK1

Becker, K. (2015, May 8). Token-Based Authentication for Web Service APIs in C# MVC .NET. Retrieved October 12, 2019, from Primary Objects: <http://www.primaryobjects.com/2015/05/08/token-based-authentication-for-web-service-apis-in-c-mvc-net/>

CodeAccess Permission.Deny Method. (n.d.). Retrieved October 12, 2019, from Microsoft: https://docs.microsoft.com/en-us/dotnet/api/system.security.codeaccesspermission.deny?view=netframework-4.8

CompressedStack Class. (n.d.). Retrieved October 12, 2019, from Microsoft: https://docs.microsoft.com/en-us/dotnet/api/system.threading.compressedstack?view=netframework-4.8

Denial of Service. (2015, February 3). Retrieved October 5, 2019, from OWASP: <https://www.owasp.org/index.php/Denial_of_Service>

Figure 1. Vulnerability Metrics. Retrieved from <https://nvd.nist.gov/vuln-metrics/cvss>.

Man-In-The-Middle Attack. (2015, August 31). Retrieved October 5, 2019, from OWASP: <https://www.owasp.org/index.php/Man-in-the-middle_attack>

(n.d.). Retrieved September 29, 2019, from Puma Scan: <https://www.pumascan.com/product/#slide-2>

NIST. (n.d.). Vulnerability Metrics. Retrieved October 4, 2019, from <https://nvd.nist.gov/vuln-metrics/cvss>.

PasswordDeriveBytes Class. (n.d.). Retrieved October 12, 2019, from Microsoft: <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.passwordderivebytes?view=netframework-4.8>

Figure 11. Critical CVE’s in the .NET Framework 4.6.2. Retrieved from <https://www.cvedetails.com/vulnerability-list.php?vendor_id=26&product_id=2002&version_id=205549&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=9&cvssscoremax=0&year=0&month=0&cweid=0&order=1&trc=7&sha=3cde49580707689fcee82492b24b4dc6596bb522>.

Figure 12. Medium CVE’s in .NET Core 2.2.7. Retrieved from <https://www.cvedetails.com/vulnerability-list.php?vendor_id=26&product_id=43007&version_id=&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=5&cvssscoremax=0&year=2018&month=0&cweid=0&order=1&trc=5&sha=a9177fe02fd1d2dec33e85326b347eb66de40a6c>.

Microsoft. (2018, December 11). CVE-2018-8540 | .NET Framework Remote Code Injection Vulnerability. Retrieved from <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8540>.

# Appendices

Appendix A: Requirements & Test Cases Tracking Spreadsheet: [Vulnerability and Requirements Tracking.xlsx](https://libertyuniv.sharepoint.com/sites/Group-CSIS481-003Fall2019CyberCapstoneI-CloudFit/_layouts/15/Doc.aspx?OR=teams&action=edit&sourcedoc=%7bE1BF4F4F-D209-4588-9B4D-20805732C13F)

Appendix B: Security Requirements: [STP - Appendix B - Security Requirements.docx](https://libertyuniv.sharepoint.com/:w:/s/Group-CSIS481-003Fall2019CyberCapstoneI-CloudFit/EaE8beJRT2tBnIyljAIz5VEB7HrTKIBt4Y4fLgzJUakcyA?e=HAHUSM)

Appendix C: Security Test Cases: [STP - Appendix C - Security Test Cases.docx](https://libertyuniv.sharepoint.com/:w:/s/Group-CSIS481-003Fall2019CyberCapstoneI-CloudFit/EX7Q7n4TfGpLogjV2MPkIn4Byk_VMkxw3lyRCEhDlgPmqw?e=tdhUfN)

Appendix D: Client Security Recommendations: [IST2 - Appendix D - Client Security Recommendations](https://libertyuniv.sharepoint.com/sites/Group-CSIS481-003Fall2019CyberCapstoneI-CloudFit/Shared%20Documents/General/Work%20In%20Progress/IST2%20-%20Appendix%20D%20-%20Client%20Security%20Recommendations.docx)

Appendix E: Data Flow Diagrams: [STP - Appendix E - Data Flow Diagrams.docx](https://libertyuniv.sharepoint.com/:w:/s/Group-CSIS481-003Fall2019CyberCapstoneI-CloudFit/EaGncPRP51JLtATjUyYoFjkBS3qJpKwHbRgaaNZr7s_m9A?e=OfqD9B)