Global Information Assurance Certification Paper

Auto-Nuke It from Orbit: A Framework for Critical Security Control Automation

GIAC (GCIH) Gold Certification

Author: Jeremiah Hainly, jhainly@gmail.com

Advisor: Adam Kliarsky

Accepted: March 13th, 2017

Abstract

Over 83% of security teams report that the use of automation in security needs to increase within the next three years (Algosec, 2016). With automation becoming a reality for a growing number of companies, there will also be an increased demand for open-sourced scripts to get started. This paper will provide a framework for prioritizing and developing security automation and will demonstrate this process by creating a script to automate a common information security response procedure – the reimaging of an infected endpoint. The primary function of the script will be to access the application program interface (API) of various enterprise software solutions to speed up the manual tasks involved in performing a reimage.

1. Introduction

The Center for Internet Security Critical Security Controls (CIS Controls) are a prioritized set of 20 recommended actions for cyber defense that provide specific and actionable ways to counter the most critical cyber-attacks (SANS, 2017). Implementing all 20 CIS Controls reduces a company's risk of cyber-attack by around 94 percent (Center, n.d.). At the core of the framework is the ability to automate these recommended actions in order to save organizations both time and money (Cole, 2016). This is rooted in the fact that implementing manual defenses to address each of the 20 CIS Controls would be financially and practically unrealistic.

When deciding which of the CIS Controls an organization should initially automate, the priority should be low-hanging fruit - or tasks that can be easily automated with little effort and high cost savings. These tasks will vary for each organization based on which CIS Controls have already been implemented and what the information security team spends the highest amount of time doing manually. An example of this would be an organization that is currently spending eight man-hours each week investigating issues related to devices of which they were previously unaware. As a result of these investigations, the security team commonly finds that the gap was caused because the server team installed a new device and did not notify their security team. In this case, the low-hanging fruit could be identified as the establishment of an automated asset inventory collection and audit process. Doing so would satisfy the first and therefore highest priority CIS Control: Inventory of Authorized and Unauthorized Devices (CIS, n.d.).

Another consideration for automation is the readiness of resources available to complete the task. For example, if a company is debating between two tasks to automate, it could be worth conducting discovery with a goal of finding open-source code, application programming interfaces (APIs), or human resources that have experience or capabilities in the automation of that task. The more resources available, the easier it will be to build, troubleshoot, and integrate automation into daily security operations. For

example, a security team with Perl scripting experience will be able to develop Perl scripts much more quickly than Python scripts. Identifying what resources and skill sets are available will make identification of manual tasks that can be automated simpler.

Once a process is identified as automatable, companies have the option to invest in vendor solutions that incorporate automation. However, the hardware, licensing, human resources, and training involved in vendor solutions makes it unrealistic to purchase a solution for every automation use case. This leads to a high demand for scripting skills in today's security market. According to McAfee's "Hacking the Skills Shortage" survey, one of largest deficits of cyber security talent is software development – with 74% of United States respondents indicating an industry scarcity (McAfee, 2016). Over 83% of security teams report that the use of automation in security needs to increase within the next three years (Algosec, 2016). With automation becoming a reality for a growing number of companies, there will also be an increased demand for open-sourced scripts to be developed.

This paper provides a framework for in-house discovery and development of security automation. As an example for utilizing the framework, a script will be written for an organization that currently has a manual process to quarantine and reimage an infected workstation – a prime example of low-hanging fruit.

2. Discovery

When a machine is infected, it is common to wipe the device's hard drive and reimage the computer with the organization's standardized image – or as a Microsoft program manager describes it – "nuke it from orbit" (Naraine, 2006). This catchphrase originates from a scene in the 1986 sci-fi film *Aliens* when the main character, Ripley, concludes that the only way to destroy the aliens inhabiting Earth is with a nuclear missile to destroy every bit of evidence that they existed. In this moment, Ripley famously says, "I say we take off and nuke the entire site from orbit. It's the only way to be sure" (Cameron, 1986). Similarly, a reimage is the only way to be sure that any

malicious software, auto-runs, or persistence mechanisms that may have been left behind on an infected device are eliminated from its hard drive.

The procedure for an information security team to "nuke" a hard drive has the potential to be repetitive, manual, and often inconsistent amongst analysts. For the company that this paper will use as an example for the proposed automation framework, the reimage process starts with the identification of an infected machine by a security analyst. After it is determined that a reimage is required, the analyst must correlate information about the computer's owner including their name and email address. The information gathered is used to communicate to the user that their computer is going to be quarantined from the network and that the client support team will reach out to him or her with the next steps. After communication is sent, the analyst must quarantine the computer using a host-based firewall solution such as Symantec Endpoint Protection (SEP). Once the quarantine is successful, the analyst must access an incident management ticketing system, such as ServiceNow, to request that the client support team back up the user's files and reimage the hard drive.

Since the process to reimage a hard drive is both common and manual for the security team in this example, the task can be qualified as a strong candidate for automation in order to allow the team to work on more novel problems. Additionally, CIS Control #3 is "Secure Configurations for Hardware and Software", making the remediation of a compromised machine a high priority to reduce risk. However, the assumption that the reimage process is low-hanging fruit for automation needs to be verified by ensuring that it is feasible for the example company to automate.

The first step in verifying the capability to automate a process is to look for open-source code from other developers who have encountered a similar issue. A few resources that can be used to identify open-source code repositories include GitHub or SourceForge – where anyone can share and edit software projects. There are numerous security practitioners, researchers, and vendors that frequently release well-developed code that can be quickly re-configured and implemented at any organization. However, if no pre-existing projects can be identified, a resource discovery process begins.

Jeremiah Hainly, jhainly@gmail.com

Resource discovery is the identification of two types of assets – technical resources and human resources. Technical resource discovery ensures that the APIs or programming modules required to automate the task are available. An API is a set of programming tools that act as building blocks for programmers to integrate their custom code into existing programs. For example, Twitter has an API available which allows developers to directly read and write Twitter data. A programming module is another building block for developers since it makes available pre-programmed functions that can be reused for many applications. Some APIs required for development of the reimage script include Splunk, SEP, and ServiceNow. An online search for the APIs for each of these products quickly returns the documentation required to automate the task (Appendix A). There is also a requirement for programming modules to support sending emails – which are available in Python via the email package and smtplib module (Python Software Foundation, n.d.). Once the technical resources have been identified, human resource discovery can begin.

The human resource discovery process includes identifying whether any existing or potential employees have the skillsets or the capability to learn the skillsets required to automate the identified task. It is also important to plan for continued support of that skillset into the future in order to maintain the infrastructure supporting the automated process. For the example company, there are multiple resources on the information security team that have Python scripting experience. Since the human and technical resources required to automate a reimage have been identified, the discovery phase ends and the planning phase begins.

3. Planning

An agile approach to software development enables scripting to be performed at the highest levels. The term agile was first coined by the *Manifesto for Agile Software Development,* which was created by 17 industry professionals that agreed on a collection of twelve principles for developing software (Beck et al., 2001). The key principles of the manifesto that apply to security automation scripting are:

1. Customer satisfaction by **early and continuous delivery** of valuable software
2. **Working software** is the principal measure of progress
3. Working software is **delivered frequently**
4. **Sustainable development**, able to maintain a constant pace
5. Continuous attention to **technical excellence** and **good design**
6. **Simplicity**—the art of maximizing the amount of work not done—is essential
7. Regularly reflect on how to **become more effective and adjust** accordingly

Ayehu, an IT Process Automation and Orchestration company, provides a crawl, walk, run approach to IT automation that supports these key principles of agile software development (Ayehu, 2016). The approach is based on the following quote from Martin Luther King, Jr.: "If you can't fly, run; if you can't run, walk; if you can't walk, crawl; but by all means keep moving" (King, 1960). While Dr. King was not automating IT and security processes, the connotation "crawl, walk, run" provides reasonable advice for the development of automation scripts. The phrase means that any problem can be tackled with a slow, calculated solution as long as the primary focus is getting it operational. As the solution is tested and proves valuable to the team, other important tasks and workflows can be added on.

The "crawl" phase of development for the reimage script includes automating a small subset of the workflow. Since there is a strong amount of Splunk API documentation available (Splunk, n.d.), the subsets of development that the "crawl" phase will focus on will be Splunk and email. The script resulting from this phase will first request user information via a Splunk query. Using the information from Splunk, the script will send the user an email to inform them that their computer was identified as infected, will be quarantined within the next hour, and that their local client support team will reach out to them soon to discuss the next steps. In the "crawl" stage, the code does not have to be perfect, but should at least be documented well so that another developer could continue into the "walk" stage without the original developer's help. At this stage,

analysts will be able to save time by automatically sending communication to the owner of the infected device.

The "walk" stage of development will add in the other important processes, including the quarantine of the targeted computer in Symantec Endpoint Protection (SEP) and the submission of a ticket in ServiceNow. SEP is similar to home anti-virus software, but has more advanced heuristic detection and firewall capabilities that allow a security team to cut off communication to the internet, or quarantine the device. ServiceNow is an Information Technology (IT) service management solution that includes an incident management ticketing system that can be leveraged to resolve incidents such as an infected computer. Connecting to each of these software solutions requires programming direct access to their respective APIs. After both features are implemented, the script will serve the primary purpose of the automation project: to quarantine and reimage an infected machine. However, the features implemented during the "walk" phase do not qualify the developer to consider the script complete.

Although the script has already proven its value, it is possible to constantly improve its usability and functionality. In terms of functionality, a key feature to improve would be getting the script to not only create new tickets, but also to update existing tickets in case an incident was already created. For usability, the script has the potential to be added to a git repository so that multiple analysts can improve the scripts. In this case, a configuration file takes passwords and proprietary data out of the script so that it does not get published on a local, and possibly insecure, git repository.

This paper will go so far as to implement editing existing ServiceNow tickets and uploading the scripts to a git repository, but there is also potential to continue growth. Additional functionalities include collecting forensic data from the target endpoint and forwarding that data to Splunk. This data could be used for insight into what trends might contribute to the infection of an organization's computers. Some questions that could be answered from logging forensic data include:

- Has this workstation been reimaged before? When?
- In which geographic location are the most computers reimaged?

Jeremiah Hainly, jhainly@gmail.com

- What processes are running on reimaged computers that aren't running on other computers?
- Are there any trends in the auto-run programs for reimaged computers?

The opportunities for this data and the potential of other data from this script are endless.

4. Crawl

The "crawl" stage of development involves initial setup of the necessary tools to begin scripting and obtaining a fundamental level of functionality from the script. For the reimage script, the level of functionality previously identified requires a connection to both Splunk and email. This is in order to lookup a user's name and email address based on his or her organizational username, then send that person an email to notify them that their device will be quarantined.

The first step for a developer to start writing the reimage script is getting a scripting language, Python, installed in their work environment. The reason Python is being used in this example is because human resources with Python skills were readily available at the example company and the modules required to complete the task were identified as available. To install Python, developers can navigate to the Python website and install the latest version of Python 2.7.X (Python Software Foundation, n.d.). After installation, the developer will need to add the program to the Windows Environment Variables as a path as seen in Figures 1 and 2.

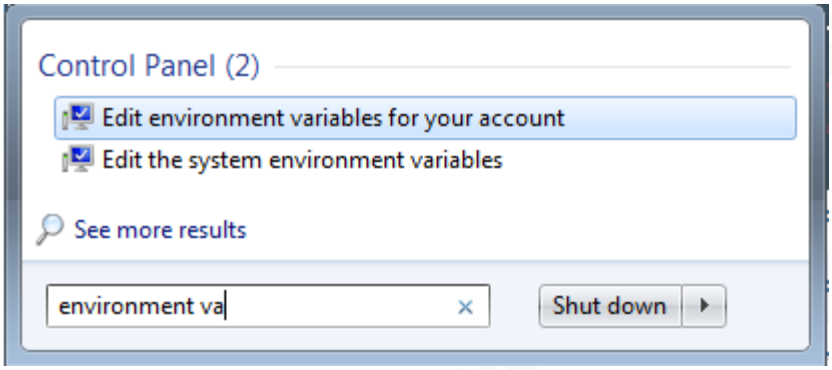

Figure 1: Environment Variables in Windows Start Menu

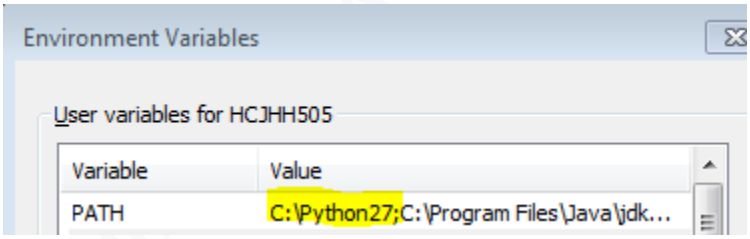

Figure 2: Python PATH variable

Variable Name: PATH

Variable Value: C:\Python27

To test that installation was successful, open your command prompt, type "Python", hit enter, type "2 + 2", and hit enter again. A result of 4 will be present if the installation is successful, as seen in Figure 3:

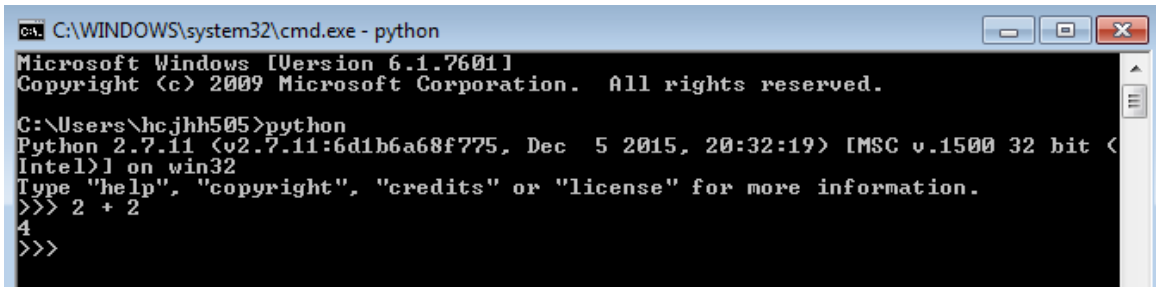

Figure 3: Testing successful Python install

The next recommended tool to install is a syntax editing program such as Sublime Text (Sublime Text, n.d.). A program that understands Python will ensure the correct syntax and formatting are used while the script is written.

Jeremiah Hainly, jhainly@gmail.com

After a tool space is successfully set up and running, the process of developing the script can begin. For the reimage script, a method must be identified to correlate user information in Splunk given only a username. Conveniently, the Active Directory database is queried daily and written to a Splunk lookup table, providing the user's name and email address, which can be correlated with a provided username. The Splunk query to generate this data from Active Directory is available in Appendix B.

Next, this script must accept user input to provide the user's login ID, authenticate to Splunk, and include the user ID within a search query that requests that person's name and email address. As identified in the discovery phase, there is a Splunk REST API that can be invoked from a Python script. Implementing the functionality involves researching example code that is available and tuning the configurations to work properly in the current environment.

Once those data values are returned, the smtplib module can be used to send the user an email. Similar to the development of the Splunk module, example code can be found online to guide the building process of sending an email with the smptlib Python module.

After testing has been conducted to prove that the script can successfully and repeatedly execute the foundational services established as goals in the discovery phase, the "crawl" phase ends and the "walk" phase begins.

5. Walk

The goal of the "walk" phase is to finish building the core functionality of the automation process into the script. The core functionality of the reimage script includes sending a quarantine command to the user's computer with Symantec Endpoint Protection (SEP) and submitting a ticket to the client support team to complete the task of reimaging the affected computer. These tasks will be the scope of the "walk" phase of development.

Submitting a ticket with ServiceNow involves calling the REST API for the application (ServiceNow, 2016). The commands for the API can be found within the platform by searching on the sidebar for "REST" as seen in Figure 4. The interface has the ability to quickly provide names for tables and columns in the incident database.

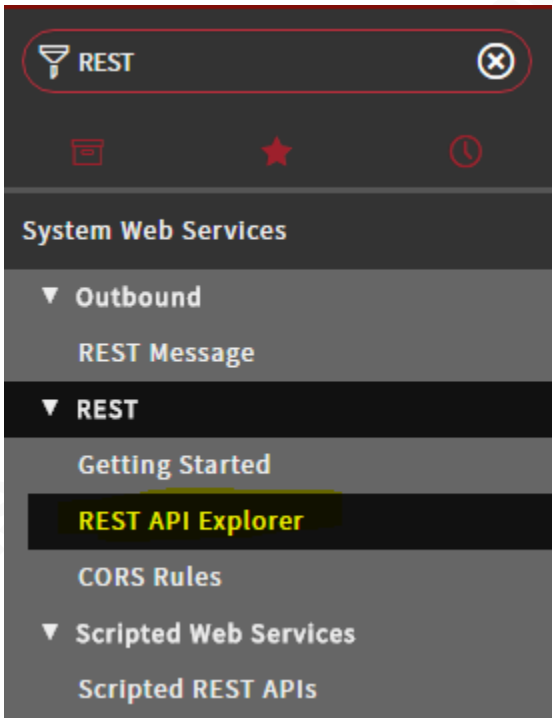

Figure 4: ServiceNow REST API Explorer

Since the ServiceNow platform available is in the company's cloud environment, a proxy will need to be configured to access outside the internal company network. Microsoft recommends using service accounts when a single person is not responsible for all of the activity conducted by that account (Microsoft, 2006). Service accounts in Active Directory are used by systems to access resources that are required to perform their primary functionalities. It is recommended to understand the implications of using service accounts and learn how to secure the accounts before implementing them into a business environment. Using service accounts will help mitigate the risk of the scripts working improperly as employees, and their account privileges with them, exit the company.

Jeremiah Hainly, jhainly@gmail.com

Connecting to Symantec Endpoint Protection is largely more ambiguous since there are not as many open-source resources available to accomplish the task. However, there is a Software Development Kit (SDK) included as part of the Symantec Endpoint Protection Manager (SEPM) installation package which includes example code to authenticate to SEPM and multiple Web Service Definition Language (WSDL) files. A WSDL is an XML document that provides specific information on which functions, inputs, and outputs are available for developers to utilize backend information for the software solution (W3C, n.d.).

After SEPM is installed, example code and the Symantec WSDL are located at the following uniform resource indicator:

\\<IPADDRESS>\d$\<SYMANTECVERSION>\Symantec_Endpoint_Protection_<SYMANTECVERSION>_Full_Installation_EN\Tools\Integration\SEPM_WebService_SDK

The first step to quarantine a computer is authenticating the script to the SEPM server. A successful authentication requires a local system account to be created on the SEPM server, which allows a user to authenticate to the SEPM Web Services interface. The Web Services interface is externally exposed on port 8446 and permits a user to create an account to authenticate to the SEPM Application Programming Interface (API). Since the API requires a password that regularly updates, a refresh token can be used in the code to retrieve a new session token each time the script is run. Details on completing this process can be found in Appendix C.

Investigating the *ClientService.WSDL* file indicates that there are two functions required to quarantine a computer given the input of the target machine's hostname: *getComputersByHostName* and *runClientCommandQuarantine*. The first command will accept the input of a hostname that is managed by SEPM, then output the associated Globally Unique Identifier (GUID) that Symantec uses to identify each SEP client. The second function accepts the GUID as input as well as a command to either "Quarantine" or "Undo" the quarantine on a target device. The Quarantine command affects the host integrity status of the computer, giving it a firewall policy that effectively disallows any communication between the device and the rest of the network. Running these commands

together will provide the goal functionality to input a hostname and directly quarantine the computer.

5.1. Testing

After completing the script, it is time to test whether it covers the scope of functionality identified during the discovery phase. Those scope items were:

- Retrieve user information from **Splunk**
- Open a ticket in **ServiceNow** to Client Support to reimage the infected device
- Send the user an **email** to notify them that their computer will be removed from the network
- Quarantine the device in **Symantec Endpoint Protection**

Figures 5 through 10 show an example of running the full reimage script, in which there are additional features already implemented from the run phase, but the core functionalities above have each been demonstrated.

The interactions depicted would be performed by a human in the loop after a detection mechanism has alerted a security analyst to a possibly infected computer. After validating that the intrusion requires remediation by way of reimaging the endpoint, the analyst would execute the compiled version of the Python script, as shown in Figure 5.

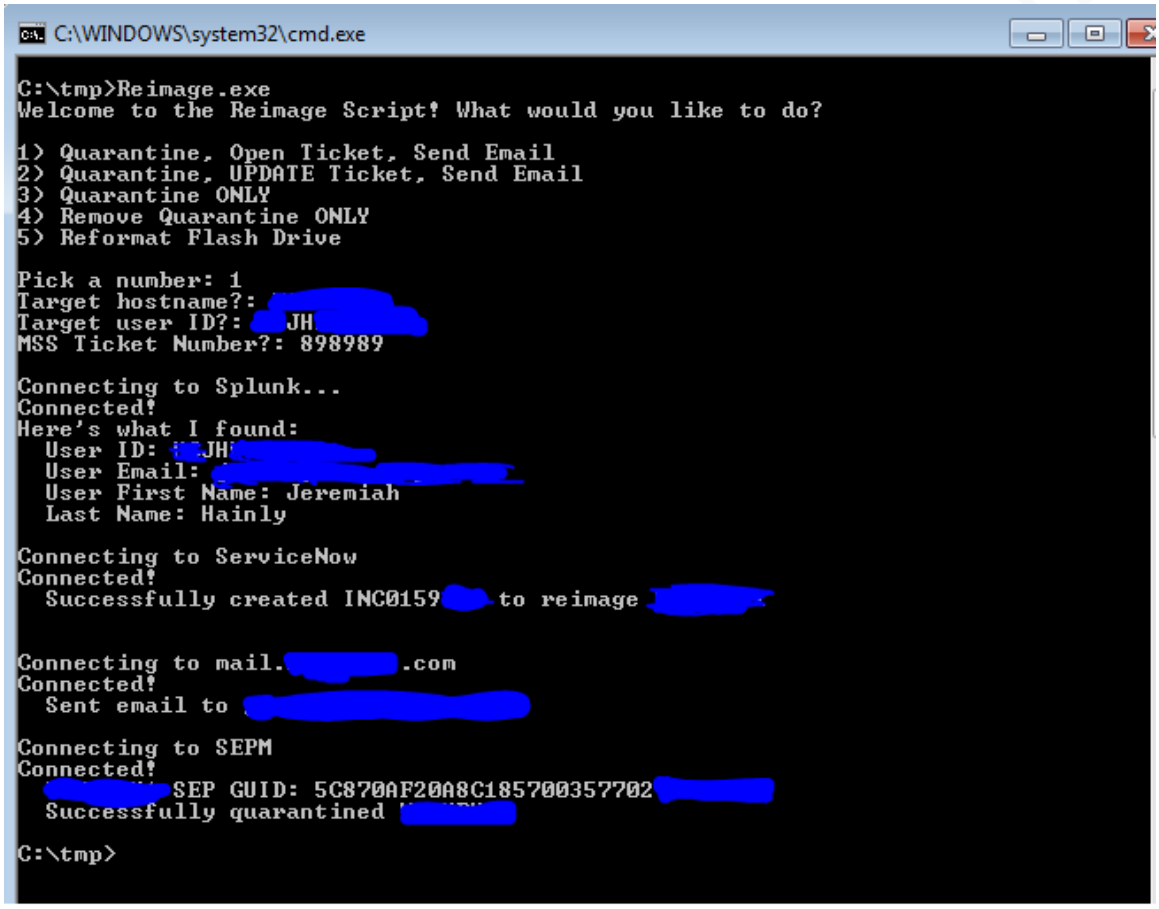

Figure 5: Running Reimage.exe from the Windows command line

The success statements in the output in Figure 5 indicate that the script ran as intended, but the success of each process can be validated by looking at the result of running the script. The first step taken by the script is to search for user information in Splunk. The query in Figure 6 requests the search history of Splunk users. In this case, the script successfully searched for the user information submitted by the user.

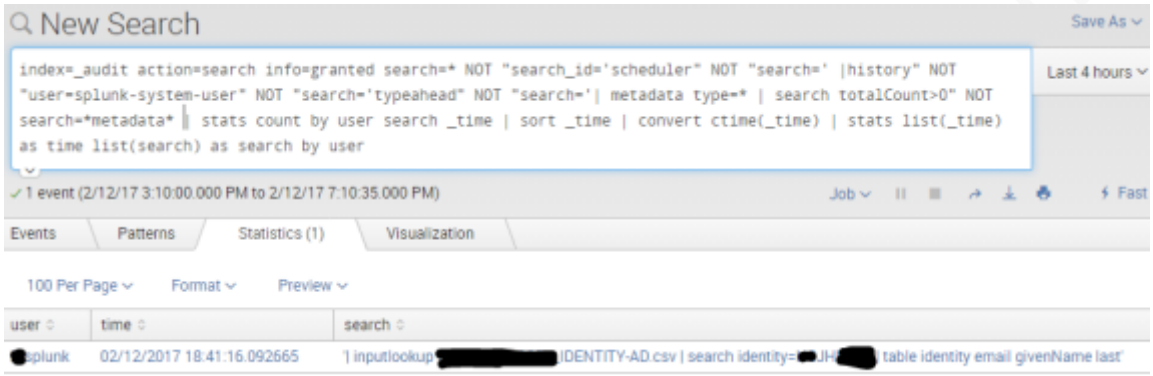

Figure 6: Search command successfully running in Splunk

The next script action that can be validated is the email that was sent to the owner of the infected workstation. Figure 7 indicates that an email was successfully sent and that the correct information was populated into the email. The information populated includes the name and email address requested from Splunk, the hostname entered in the command prompt by the security analyst, and the ServiceNow ticket number that was generated as a result of running the script.

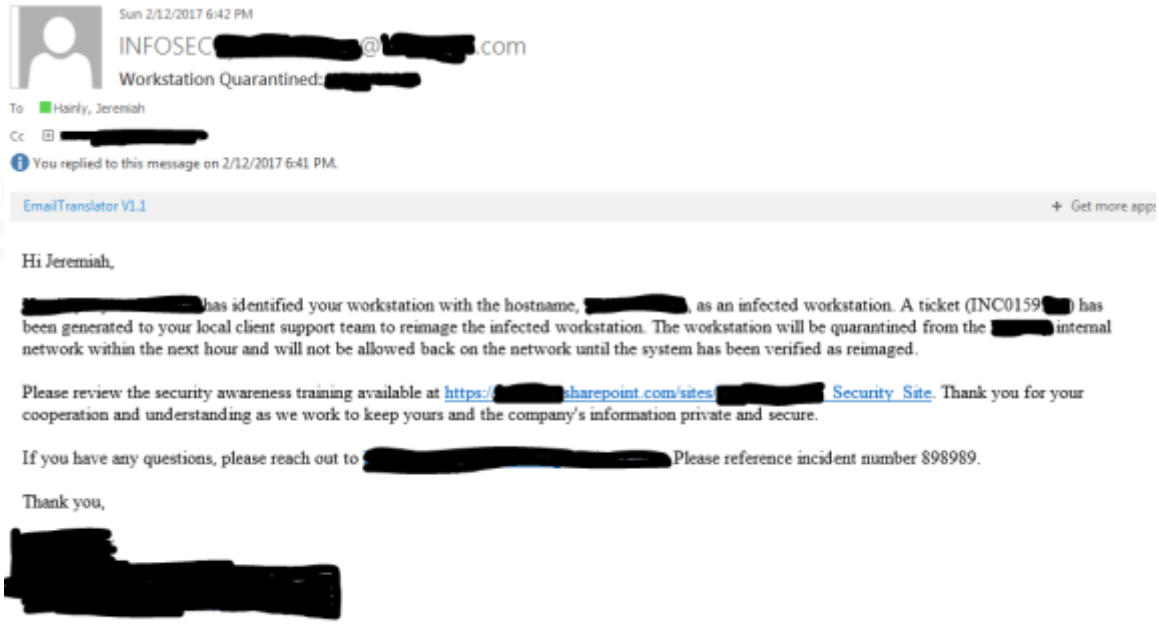

Figure 7: Email successfully sent to affected user

Jeremiah Hainly, jhainly@gmail.com

Next, the script accesses the ServiceNow API to create an incident based on the information provided by the analyst. Figures 8 and 9 show that the generated incident contains the parameters defined by the script in order to open an incident.

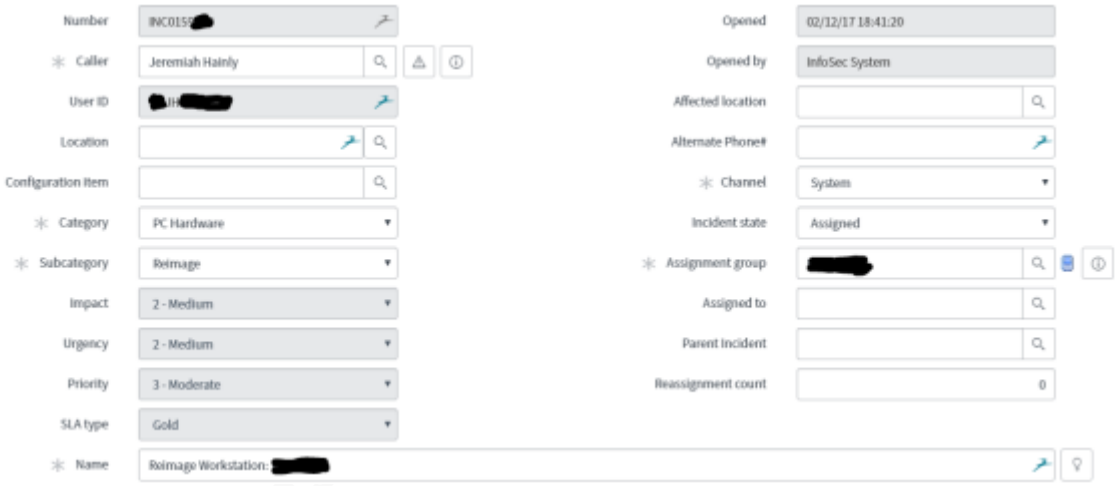

Figure 8: ServiceNow ticket successfully opened

Figure 9: ServiceNow ticket details successfully submitted

Lastly, the reimage script contacts the SEP SOAP API using the system account designated in the script and invokes a quarantine command. The results from running the command can be viewed in SEP Management console which shows that the host's integrity status failed was quarantined by the administrator (Figure 10).

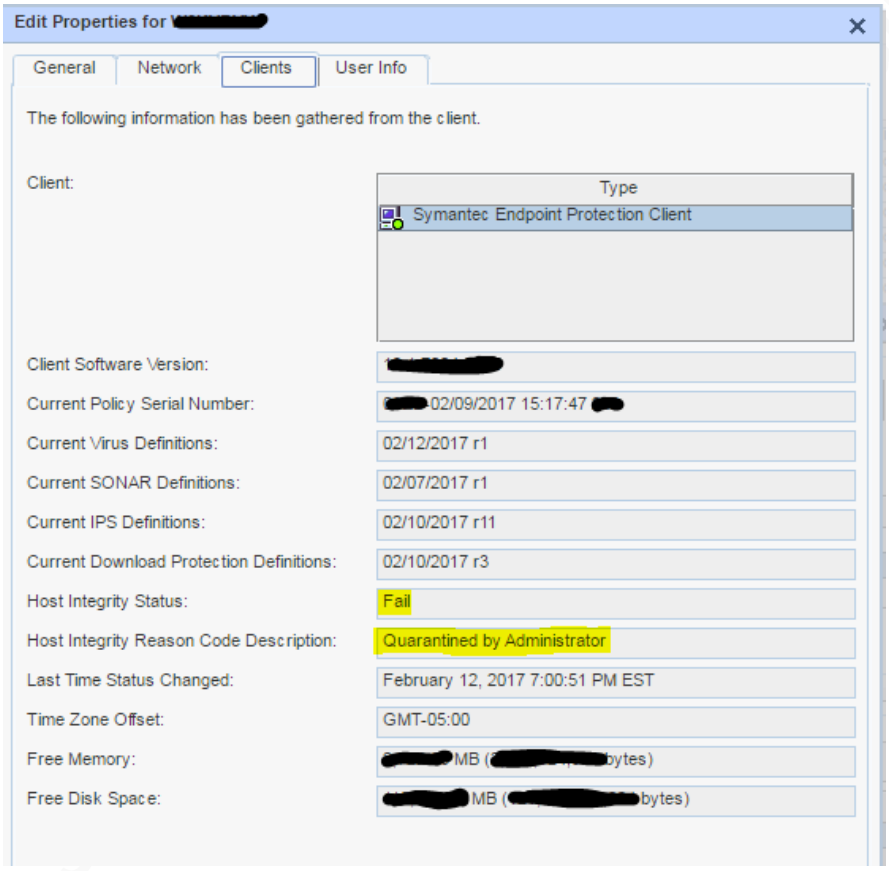

Figure 10: Symantec Endpoint Protection quarantine successfully imposed

After testing and successfully implementing the core functionalities, the "walk" phase completes and the "run" phase can begin.

6. Run

According to Ayehu's "crawl, walk run" development methodology, the "run" phase of automation development is the time to extend automation to new initiatives and applications (2016). In the context of security automation, there are many opportunities contained in the CIS Controls themselves.

CIS Control #3 is "Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers" (SANS, 2017). Since automation development is creating software that will run on company-owned devices, it is critical to ensure that the software is protected against an attack. A common mistake by software

developers, even at large security companies like Cisco, is hard-coding passwords (Chirgwin, 2016). A hard-coded password is one that is written in plain text rather than encrypted or stored in a separate, secure location. Not only does this allow project developers to view the password, it is also difficult to change the password without patching the software. This forces developers and system owners to choose between confidentiality of system passwords and the availability of the automation script (OWASP, 2016).

While there are multiple solutions to removing hard-coded passwords, one of the simplest methods is to create a configuration file for the script to reference. ConfigParser is a Python class that implements a basic configuration file parser language, which allows the script to reference an external file for sensitive data like passwords and sensitive company information (Python, 2017).

The requirement for a configuration file can be illustrated in the context of the reimage script, especially if the code is stored in both a public and private git repository. The purpose of sharing the code is so that additional developers, both inside and outside of the company, can make changes to improve the automation scripts. However, making code publically available also exposes any system account passwords that may have been hard-coded previously. A configuration file combined with the Python ConfigParser class provides an opportunity to remove those passwords. Since the code is stored in a git repository, it is simple to add the configuration file to the .gitignore file so that sensitive information is not published to the repository.

There are a few other features included in the publically available version of the reimage script. The first is the ability to request that a computer is reimaged by editing a ServiceNow ticket that has already been opened. The only difference is that instead of using an HTTP POST method, the script uses an HTTP PUT method. According to ServiceNow's REST API documentation, this provides an update to a ticket instead of creating a new ticket (ServiceNow, 2015).

Next, there is a feature to remove a Symantec Endpoint Protection quarantine from a computer. This is useful in a situation where a quarantine was originally placed on

an incorrect computer or the issue was resolved without ultimately requiring a reimage. As a note, a computer that is reimaged will automatically remove the quarantine because SEP registers the device with a new GUID and eventually removes the old device from its database.

Lastly, the script has the ability to quarantine a device in Symantec Endpoint Protection without sending an email or opening a ServiceNow ticket. This feature can be useful in the case that the script originally failed to quarantine the device or there is no need to open a ticket for reimaging the infected machine. The full code for the reimage script can be found in Appendix D and is also available online on GitHub (Hainly, 2017).

Additional improvement opportunities in the context of the reimage script are endless and not necessarily demonstrated in the publically available code or in this paper. For example, reimaging a computer may only be a small part of the reimage process for many organizations. Other steps may include a forensic investigation to identify a root cause for the infection of the device. Following the discovery phase of searching for open-source scripts, an online search returns multiple instances of automated forensics tools such as Brian Moran's Live Response Collection scripts. These batch files trigger a series of forensics tools that collect data from target endpoints. Running the tools on every reimaged endpoint could provide security analysts with a large data set which could be logged into Splunk or Hadoop so that a security data analytics team could understand the commonalities between infected machines.

7. Conclusion

The goal of this research is to propose a framework by which manual processes performed by information security teams could be automated using the Critical Security Controls for prioritization of tasks that can be automated with little effort and high cost savings – or low-hanging fruit. Several practical lessons may be drawn from the research through the demonstration of the framework's application in creating a Python script to reimage a computer that was identified as infected. The research contributes to the incident response body of knowledge by applying the "crawl, walk, run" approach to

software development and by providing an open-source tool to reimage infected endpoints – or "nuke it from orbit."

Jeremiah Hainly, jhainly@gmail.com

References

Algosec. (2016). The State of Automation in Security (Rep.). doi: https://www.algosec.com/wp-content/uploads/2016/03/The-State-of-Automation-in-Security-Survey-Final.pdf

Ayehu. (2016, July 04). The Crawl, Walk, Run Approach Making IT Process Automation Work for You. Retrieved February 12, 2017, from http://ayehu.com/the-crawl-walk-run-approach-to-making-it-process-automation-work-for-you/

Beck, K., Grenning, J., Martin, R., Beedle, M., Highsmith, J., Mellor, S., . . . Marick, B. (2001). Principles behind the Agile Manifesto. Retrieved February 12, 2017, from http://agilemanifesto.org/principles.html

Cameron, J. (Director). (1986). Aliens [Motion picture on DVD]. United States: 20th Century Fox.

Center for Internet Security. (n.d.). CIS Controls. Retrieved March 05, 2017, from https://www.cisecurity.org/critical-controls.cfm

Chirgwin, R. (2016, January 13). Cisco admins gear up for a late night – hardcoded password in wireless points nuked. Retrieved February 12, 2017, from https://www.theregister.co.uk/2016/01/13/cisco_admins_gear_up_for_a_late_night/

Cole, E. (2016, August 18). SANS 401 Security Essentials. Lecture presented at SANS OnDemand.

Hainly, J. (2017, January 11). Reimage. Retrieved February 17, 2017, from https://github.com/jhainly/Reimage

King, M. L., Jr. (1960, April 10). Keep Moving from This Mountain. Speech presented at Founder's Day Address in Spelman College. Retrieved February 12, 2017, from https://swap.stanford.edu/20141218225553/http:/mlk-kpp01.stanford.edu/primarydocuments/Vol5/10Apr1960_KeepMovingfromThisMountain,AddressatSpelmanCollege.pdf

Jeremiah Hainly, jhainly@gmail.com

McAfee. (2016). Hacking the Skills Shortage (p. 21, Tech.). Santa Clara, CA: McAfee. doi: https://www.mcafee.com/us/resources/reports/rp-hacking-skills-shortage.pdf

Microsoft. (2006, September 26). Securing Critical and Service Accounts. Retrieved February 23, 2017, from https://msdn.microsoft.com/en-us/library/cc875826.aspx

Naraine, R. (2006, April 4). Microsoft Says Recovery from Malware Becoming Impossible. Retrieved March 05, 2017, from http://www.eweek.com/c/a/Security/Microsoft-Says-Recovery-from-Malware-Becoming-Impossible

OWASP. (2016, February 14). Password Management: Hardcoded Password. Retrieved February 12, 2017, from https://www.owasp.org/index.php/Password_Management:_Hardcoded_Password

Python Software Foundation. (n.d.). Download Python. Retrieved February 12, 2017, from https://www.Python.org/downloads/

Python Software Foundation. (n.d.). ConfigParser — Configuration file parser. Retrieved February 12, 2017, from https://docs.Python.org/2/library/configparser.html

Python Software Foundation. (2017). Smtplib — SMTP protocol client. Retrieved February 12, 2017, from https://docs.Python.org/2/library/smtplib.html

ServiceNow. (2016, February 08). REST API. Retrieved February 12, 2017, from http://wiki.servicenow.com/index.php?title=REST_API#gsc.tab=0

ServiceNow. (2015, October 06). Getting Started with REST. Retrieved February 17, 2017, from http://wiki.servicenow.com/index.php?title=Getting_Started_with_REST#gsc.tab=0

SANS. (n.d.). CIS Critical Security Controls. Retrieved February 12, 2017, from https://www.sans.org/critical-security-controls

SANS. (n.d.). CIS Critical Security Controls: A Brief History. Retrieved from SANS: https://www.sans.org/critical-security-controls/history

Jeremiah Hainly, jhainly@gmail.com

Splunk. (2017). REST API Reference Manual. Retrieved February 12, 2017, from http://docs.splunk.com/Documentation/Splunk/6.5.1/RESTREF/RESTprolog

Sublime Text. (n.d.). Sublime Text. Retrieved February 12, 2017, from https://www.sublimetext.com/

W3C. (n.d.). Web Services Description Language (WSDL). Retrieved February 12, 2017, from https://www.w3.org/TR/wsdl

Jeremiah Hainly, jhainly@gmail.com

Appendix A

The following are examples of searching for the appropriate API resources to complete the reimage script. Usually, a search of the technology name followed by "API" is sufficient to find the appropriate resource. When using a specific scripting language, such as Python, it can be useful to add the name of the language to the search engine query.

Splunk

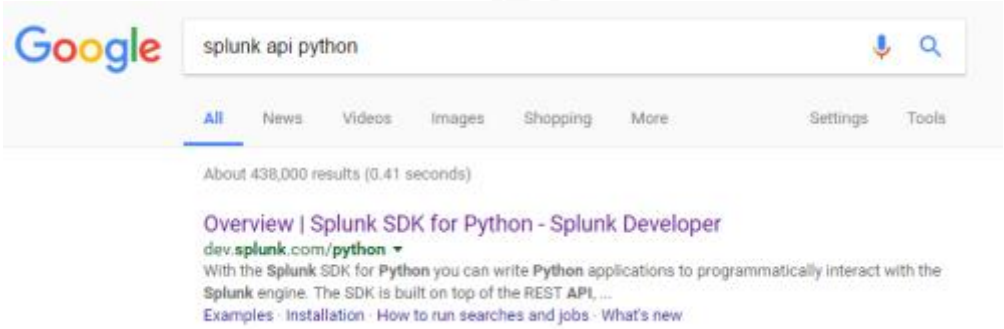

Figure 11: Google Search for Splunk API

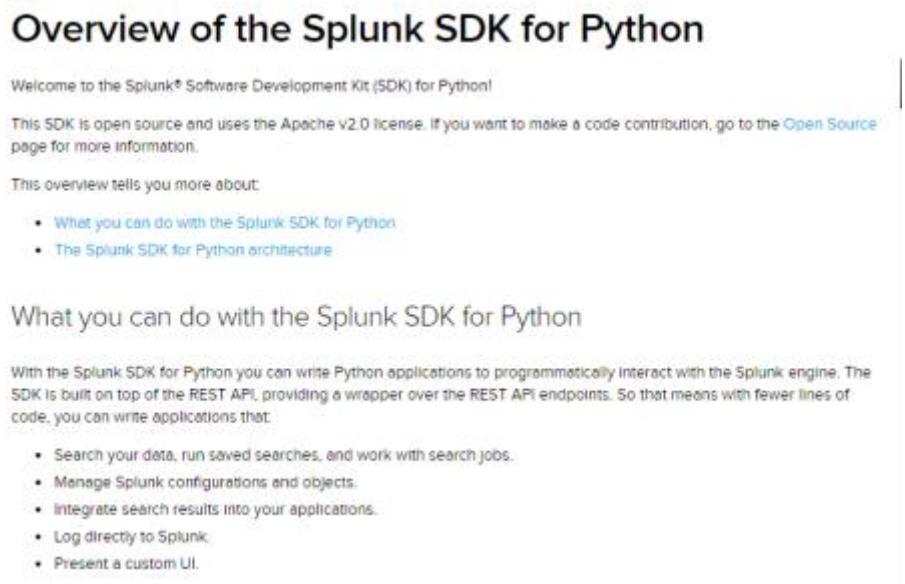

Figure 12: Quick look into the results from the search

Jeremiah Hainly, jhainly@gmail.com

Symantec Endpoint Protection

Note: There is very little documentation available directly online for SEP. It is recommended to access the files included in the installation package for SEP to have detailed information. To reiterate from the main content of the paper, the link to access documentation is:

\\<IPADDRESS>\d$\<SYMANTECVERSION>\Symantec_Endpoint_Protection_<SYMANTECVERSION>_Full_Installation_EN\Tools\Integration\SEPM_WebService_SDK

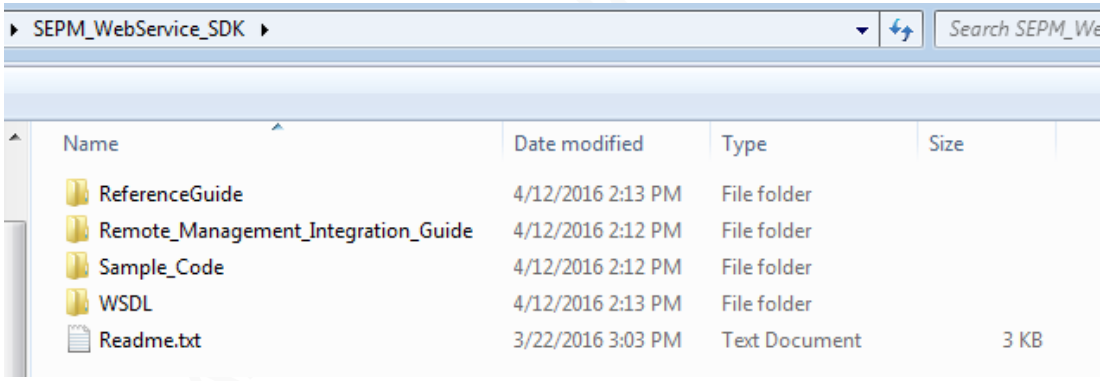

Figure 13: Contents of the path provided above

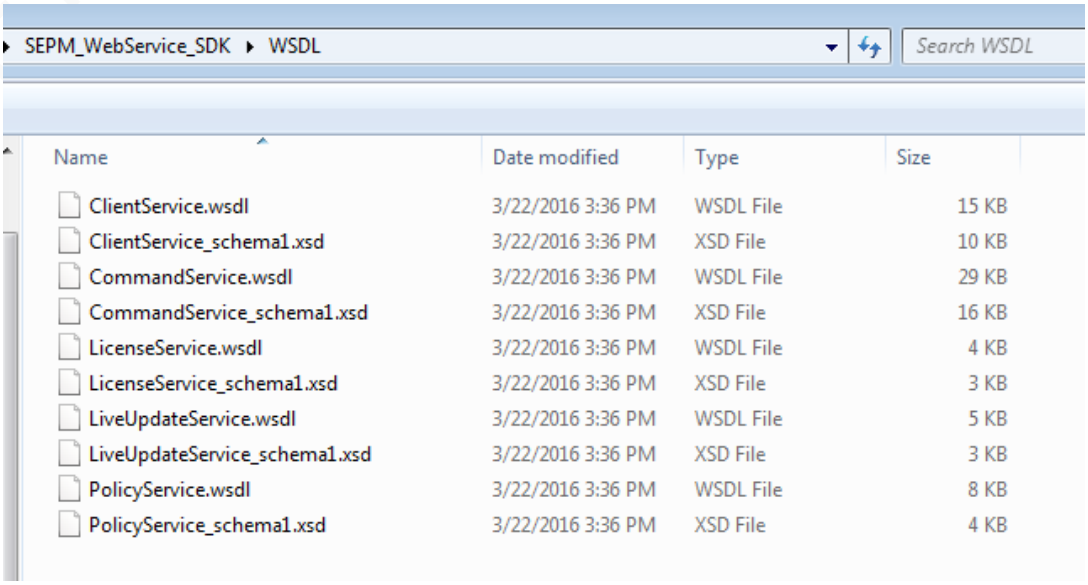

Figure 14: Contents of the WSDL folder

Jeremiah Hainly, jhainly@gmail.com

ServiceNow

There is strong documentation available online for the ServiceNow API, but the REST API tool within each ServiceNow deployment will provide the exact database schema for the incident table. Every deployment of ServiceNow is different, so it is important to follow the appropriate schema.

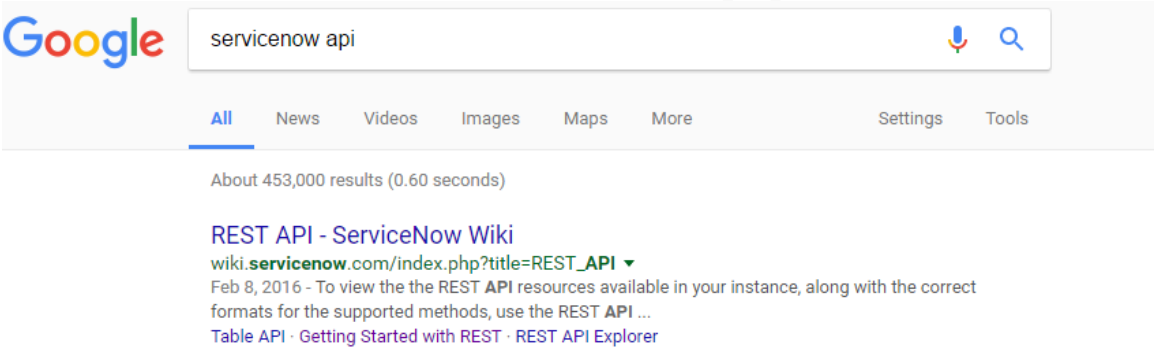

Figure 15: Google search for ServiceNow API bring back results including REST API Explorer

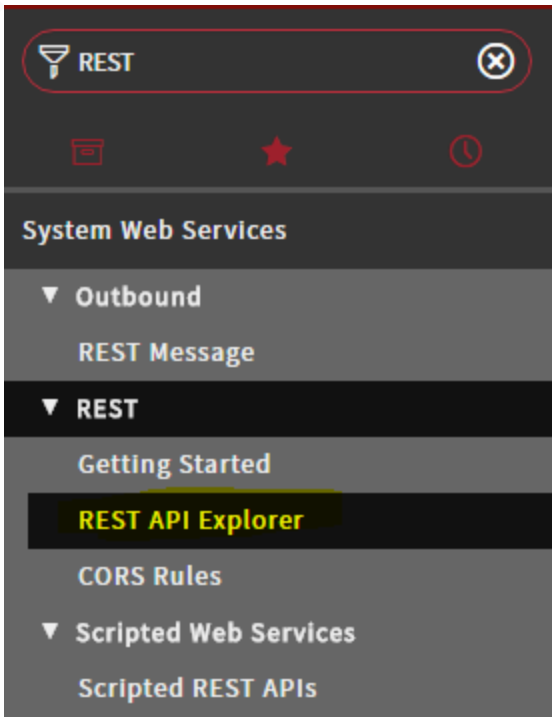

Figure 16: Searching for the REST API Explorer

Jeremiah Hainly, jhainly@gmail.com

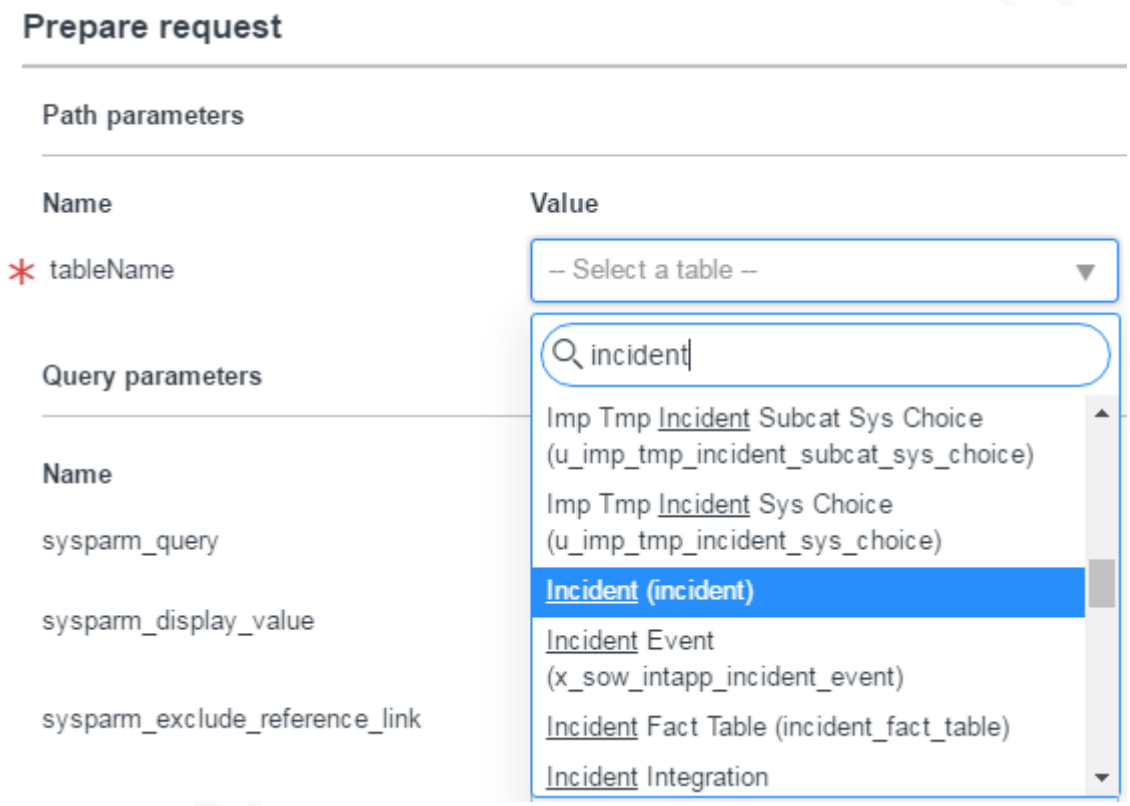

Figure 17: Searching for the incident table

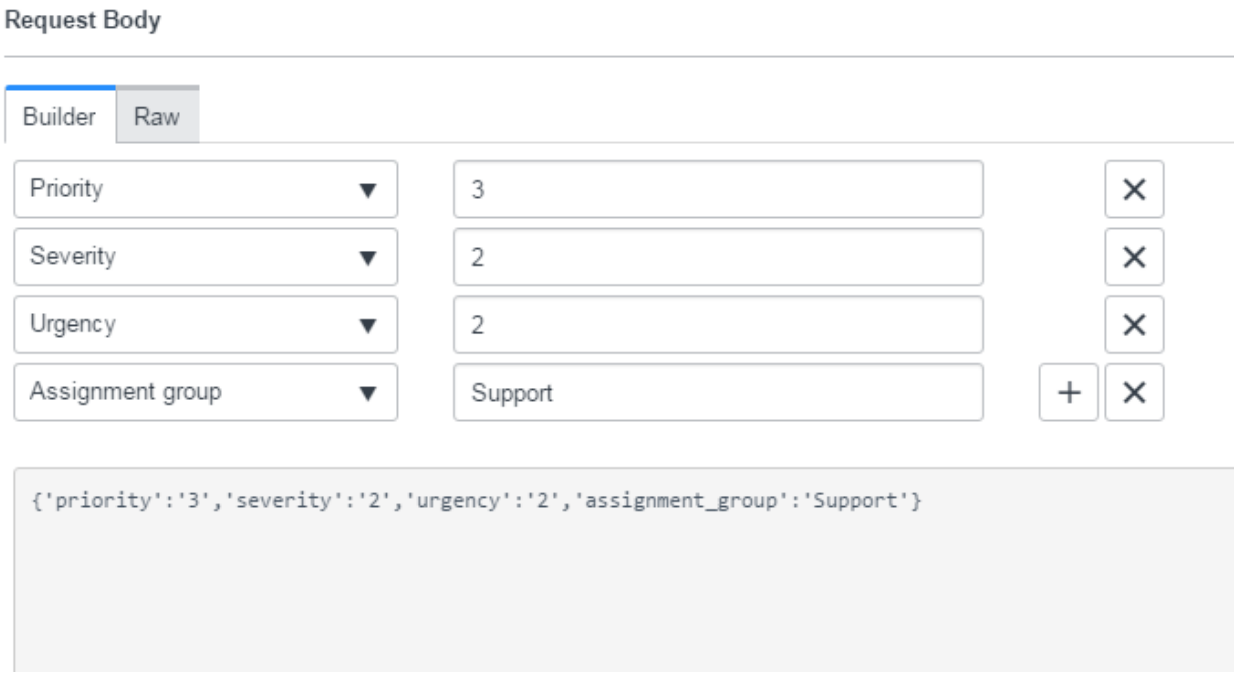

Figure 18: Building a POST query with the API explorer

Jeremiah Hainly, jhainly@gmail.com

Appendix B

Splunk Query to Active Directory Database

The following query uses the LDAP protocol to access Active Directory information. The requested fields are over-written to a lookup table on a daily basis in order to maintain its integrity.

*| ldapsearch domain=**<DOMAIN NAME>** search = "(&(objectclass=user)(!(objectClass=computer)))" | search userAccountControl = "*NORMAL_ACCOUNT*" |eval suffix=""|eval priority="medium"|eval category="normal"|eval watchlist="false"|eval endDate="" |table sAMAccountName , personalTitle , displayName , givenName , sn , suffix , mail , telephoneNumber , mobile , title , manager , priority , department , category , watchlist , whenCreated , endDate , memberOf , lastLogonTimestamp , pwdLastSet , sAMAccountName , dn , userAccountControl , description , company , accountExpires , extensionAttribute9 , countryCode , userAccountControl , c |rename sAMAccountName as identity, personalTitle as prefix, displayName as nick, givenName as first, sn as last, mail as email, telephoneNumber as phone, mobile as phone2, manager as managedBy, department as bunit, whenCreated as startDate, extensionAttribute9 as workStatus, countryCode as country, userAccountControl as enabled | outputlookup **<SPLUNK LOOKUP NAME>.**csv*

Jeremiah Hainly, jhainly@gmail.com

Appendix C

This could easily be the most difficult part of this scripting exercise. The process below indicates how to access Symantec's Web Services API without the need to manually get a refresh token when the previous token expires. This is very important in order to maintain reasonable usability on an information security team.

Creating a SEP System Account for API Access

1. Navigate to https://<SEPMHOSTNAME>:8443/console/apps/sepm
2. Login
3. Admin > Add an administrator
4. Username: <accountname>-local
5. Full Name: Quarantine Account
6. Email Address: Security distribution list

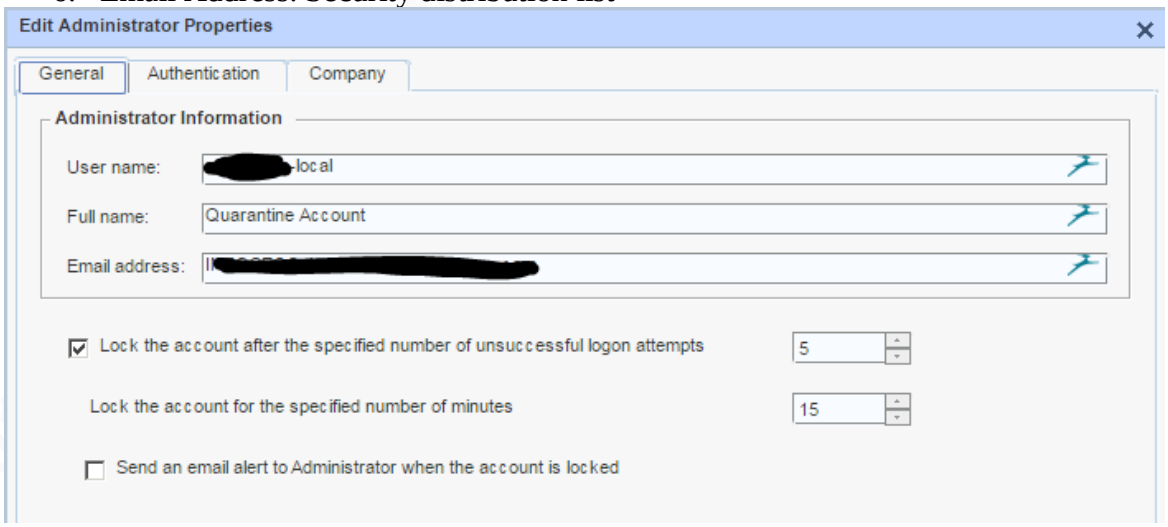

Figure 19: Example of administrator configurations

7. Authentication tab: Symantec Endpoint Protection Manager Authentication

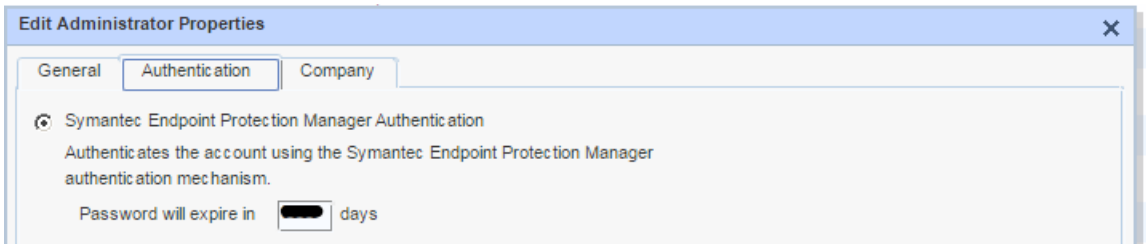

Figure 20: Local Authentication Options of SEPM

8. OK

Jeremiah Hainly, jhainly@gmail.com

9. Navigate to https://<SEPMHOSTNAME>:8446/sepm/oauth/viewClientApps.do
10. Login with the local account previously created
11. Add an application
12. Provide a name
    - <DOMAIN>/<USER ID OF PERSON CREATING THIS>:web
13. Navigate to https://<SEPMHOSTNAME>:8446/sepm/oauth/authorize?response_type=code&client_id=<CLIENT_ID_FROM_LAST_STEP>&redirect_uri=https://<SEPMHOSTNAME>:8443/sepm
14. Click Authorize

Figure 21: Authorizing access to SEPM web services on port 8443

15. Check the URL. The format will be https://<SEPMHOSTNAME>:8443/sepm?code=<COPYTHISCODE>
16. Copy that code

Figure 22: Example of SEPM code

17. Navigate to https://<SEPMHOSTNAME>:8446/sepm/oauth/token?grant_type=authorization_code&client_id=<CLIENTID>&client_secret=<CLIENTSECRET>&redirect_uri=https://<SEPMHOSTNAME>:8443/sepm&code=<CODEFROMBEFORE>

Jeremiah Hainly, jhainly@gmail.com

18. The page should return a dictionary of key value pairs, including the value of the refreshToken
    - {"value":"<VALUE>","expiration":1487419737468,"tokenType":"bearer","refreshToken":{"value":"<COPYTHISVALUE>","expiration":1518912537437},"scope":[],"additionalInformation":{},"expired":false,"expiresIn":43199}
19. Copy the value of the refresh token and save it somewhere for later in your code. If lost, return to step 13 and repeat
20. The final string to put into your script will be https://<SEPMHOSTNAME>:8446/sepm/oauth/token?grant_type=refresh_token&client_id=<CLIENTID>&client_secret=<CLIENTSECRET>&redirect_uri=https://<SEPMHOSTNAME>:8443/sepm&refresh_token=<REFRESHTOKEN>

Appendix D

The full script can be found online here: https://github.com/jhainly/Reimage

README.md

1.1. Synopsis

Reimage is a Python script that provides information technologists with a template for automating a task between **Splunk**, **ServiceNow**, **Symantec Endpoint Protection (SEP)**, and **email**.

1.2. Getting Started

Change the configs.template file contents to match your environment. Then, rename the file to configs.ini.

Either run the raw Python or use pyinstaller on the provided spec file to compile into an executable.

```
pyinstaller.exe --onefile Reimage.spec
```

1.3. Code Example

```
C:\Users\Me\Downloads>Reimage.exe
Welcome to the Reimage Script! What would you like to do?

1) Quarantine, Open Ticket, Send Email
2) Quarantine, UPDATE Ticket, Send Email
3) Quarantine ONLY
4) Remove Quarantine ONLY
5) Reformat Flash Drive

Pick a number:
```

1.4. Motivation

The use case in this script is an information security team that identifies a computer that requires a reimage. This script automates the normally manual

Jeremiah Hainly, jhainly@gmail.com

process to lookup user information in **Splunk**, quarantine the affected computer in **SEP**, submit a ticket to their client support to request a reimage in **ServiceNow**, and send an **email** to the owner of the computer.

1.5. About the Author

Jeremiah Hainly: https://www.linkedin.com/in/jeremiahhainly

1.6. Special Thanks

Brian Nafziger: https://www.linkedin.com/in/bnafziger

Reimage.py

This is the primary Python file that invokes the rest of the files contained in this package. It begins by asking the user for input, then references the appropriate scripts to complete the requested task(s).

```
# Purpose: Submit host for reimage via ServiceNow, send them an email, and quarantine
the host in SEPM


import sys
import time
import splunkReimage
import servicenowReimage
import emailReimage
import sepReimage


def main():
	userOption = input("Welcome to the Reimage Script! What would you like to
do?\n\n1) Quarantine, Open Ticket, Send Email\n2) Quarantine, UPDATE Ticket, Send
Email\n3) Quarantine ONLY\n4) Remove Quarantine ONLY\n5) Reformat Flash Drive\n\nPick a
number: ")


	if userOption == 1:
		sepCommand = "Quarantine"
		hostname = raw_input("Target hostname?: ")
		reimageUser = raw_input("Target user ID?: ")
		mssTicket = raw_input("MSS Ticket Number?: ")
		userID, userEmail, firstName, lastName =
splunkReimage.search(reimageUser)
		snTicket = servicenowReimage.submit(hostname, reimageUser)
		emailReimage.reimage(hostname, firstName, userEmail, mssTicket,
snTicket)
		sepReimage.reimage(hostname, sepCommand)
	elif userOption == 2:
		sepCommand = "Quarantine"
		snTicket = raw_input("ServiceNow Ticket Number? (Number only): ")
		hostname = raw_input("Target hostname?: ")
		reimageUser = raw_input("Target user ID?: ")
```

Jeremiah Hainly, jhainly@gmail.com

```
                mssTicket = raw_input("MSS Ticket Number?: ")
                userID, userEmail, firstName, lastName =
splunkReimage.search(reimageUser)
                sysID = servicenowReimage.request(snTicket)
                servicenowReimage.update(sysID, hostname, reimageUser)
                emailReimage.reimage(hostname, firstName, userEmail, mssTicket,
snTicket)
                sepReimage.reimage(hostname, sepCommand)
        elif userOption == 3:
                sepCommand = "Quarantine"
                hostname = raw_input("Target hostname?: ")
                sepReimage.reimage(hostname, sepCommand)
        elif userOption == 4:
                sepCommand = "Undo"
                hostname = raw_input("Target hostname?: ")
                sepReimage.reimage(hostname, sepCommand)
        elif userOption == 5:
                hostname = raw_input("Target hostname?: ")
                reimageUser = raw_input("Target user ID?: ")
                mssTicket = raw_input("MSS Ticket Number?: ")
                userID, userEmail, firstName, lastName =
splunkReimage.search(reimageUser)
                emailReimage.reformat(hostname, firstName, userEmail, mssTicket)
        else:
                print "Invalid input."
                sys.quit()
main()
```

Reimage.spec

The spec file is used to compile the scripts into an executable version of the code. This can be done using pyinstaller via the command line:

pyinstaller --onefile Reimage.spec

```
block_cipher = None
a = Analysis(['Reimage.py'],
             pathex=['.\\'],
             binaries=None,
             datas=None,
             hiddenimports=[],
             hookspath=[],
             runtime_hooks=[],
             excludes=[],
             win_no_prefer_redirects=False,
             win_private_assemblies=False,
             cipher=block_cipher)


a.datas += [('smallCD.png','smallCD.png','DATA'),('configs.ini','configs.ini','DATA')]


pyz = PYZ(a.pure, a.zipped_data,
             cipher=block_cipher)
exe = EXE(pyz,
          a.scripts,
          a.binaries,
          a.zipfiles,
          a.datas,
          name='Reimage',
          debug=False,
          strip=False,
          upx=True,
          console=True )
```

Configs.template

This file provides a template for a company to employ the same script in their environment. These variables are what will change for every organization. Once the data is populated, rename the file to "configs.ini".

```
proxytype = # HTTPS or HTTP proxy? [https|http]
proxyurl = https://<proxy username>:<proxy password>@<proxy hostname/IP>:<proxy port>


[production_servicenow]
url = https://<hostname>.service-now.com/api/now/table/incident
user = # ServiceNow production username
pwd = # ServiceNow production password


[development_servicenow]
url = https://<hostname>.service-now.com/api/now/table/incident
user = # ServiceNow dev username
pwd = # ServiceNow dev password


[snOptions]
teamName = # Name of security team (Hershey Information Security)
assignment_group = # Incident assignment group value
impact = # Incident impact value
urgency = # Incident urgency value
priority = # Incident priority value
incident_state = # Incident state value
state = # Incident state value
category = # ServiceNow category
subcategory = # ServiceNow sub-category


[splunk]
host= # Splunk hostname or IP address
port= # Splunk port (default is 8089)
username= # Splunk system account username
password= # Splunk system account password
lookupFile = # Splunk active directory lookup file
```

Jeremiah Hainly, jhainly@gmail.com

```
[sep]
authurl =
https://<hostname>:<port>/sepm/oauth/token?grant_type=refresh_token&client_id=<your-
client-id>&client_secret=<your-client-
secret>&redirect_uri=https://localhost/sepm&refresh_token=<your-refresh-token>
wsdl = https://<hostname>:<port>/sepm/ws/v1/ClientService?wsdl


[email]
testEmail = # Email address for person testing scripts
groupEmail = # Name of distribution email address for team
smtp = # Name of SMTP mail server
trainingSite = # Name of training website
teamName = # Name of security team (Hershey Information Security)
teamLogo = # filename of image to include in signature of emails
```

Jeremiah Hainly, jhainly@gmail.com

emailReimage.py

```
# Purpose: Send emails to users to notify them of a reimage. This is part of a larger
script
#          that verifies the user's identity in Splunk and sends the user an email
#          to notify of a reimage ticket being created.


######## Imports ##########
# Import required modules #
###########################
import smtplib
import sys
import os
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
from email.MIMEImage import MIMEImage
import ConfigParser


#################### RESOURCE PATH ######################
# Changes the resource path so that the image in the #####
# email can be referenced when compiled with pyinstaller #
# but also referenced when running in Python #############
#########################################################
def resource_path(relative_path):
    # Get absolute path to resource, works for dev and for PyInstaller
    try:
              # PyInstaller creates a temp folder and stores path in _MEIPASS
              base_path = sys._MEIPASS
    except Exception:
              base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)


# Define parser for configuration file
parser = ConfigParser.RawConfigParser()
parser.read(resource_path('configs.ini'))


################ METHOD 1 ################################
# Send email to user to reformat an infected flash drive #
```

Jeremiah Hainly, jhainly@gmail.com

```
#########################################################
def reformat(targetHost, targetFirst, targetEmail, mssNumber):
        print "\nConnecting to " + parser.get('email', 'smtp')
        try:
                # Set email from, to, cc
                strFrom = parser.get('email', 'groupEmail')
                strTo = targetEmail
                #strCc = parser.get('email', 'testEmail')   # TEST
                strCc = parser.get('email', 'groupEmail')   # PRODUCTION


                # Create the root message and fill in the from, to, and subject headers
                msgRoot = MIMEMultipart('related')
                msgRoot['Subject'] = 'Flash Drive Reformat'
                msgRoot['From'] = strFrom
                msgRoot['To'] = strTo
                msgRoot['Cc'] = strCc
                msgRoot.preamble = 'This is a multi-part message in MIME format.'


                # Encapsulate the plain and HTML versions of the message body in an
                # 'alternative' part, so message agents can decide which they want to
display.
                msgAlternative = MIMEMultipart('alternative')
                msgRoot.attach(msgAlternative)


                msgText = MIMEText('This is the alternative plain text message. Error
with HTML version')
                msgAlternative.attach(msgText)


                # Reference the image in the IMG SRC attribute by the ID we give it
below
                msgText = MIMEText("""\
                        <!doctype html5>
                        <html>
                        <body>
                                <p>Hi """ + targetFirst + """,</p>
                                <p>""" + parser.get('email','teamName') + """ has
identified that a flash drive with malicious files was plugged into your Hershey
computer with the hostname: """ + targetHost + """. Symantec blocked the files from
```

```
copying to your computer, but did not clean the flash drive.</p>
                        <p>We recommend formatting your flash drive, which will
erase ALL files (including hidden files) from the drive. You can format your flash
drive by following these steps:</p>
                        <ul>
                                <li>Click "Start" > "Computer"</li>
                                <li>Right click on flash drive</li>
                                <li>Click "Format..."</li>
                                <li>File system: NTFS</li>
                                <li>Click "Start"</li>
                        </ul>
                        <p>Thank you for your cooperation and understanding as we
work to keep yours and the company's information private and secure. If you have any
questions, please reach out to """ + parser.get('email', 'groupEmail') + """. Please
reference MSS Incident ID #""" + mssNumber + """.</p>
                        <p>Thank you,</p>
                        <img src="cid:image1">
                </body>
                </html>
                """, 'html')
        msgAlternative.attach(msgText)


        # This example assumes the image is in the current directory
        fp = open(resource_path('smallCD.png'), 'rb')
        msgImage = MIMEImage(fp.read())
        fp.close()


        # Define the image's ID as referenced above
        msgImage.add_header('Content-ID', '<image1>')
        msgRoot.attach(msgImage)


        # Send the email (assumes SMTP authentication is not required)
        import smtplib
        smtp = smtplib.SMTP()
        smtp.connect(parser.get('email', 'smtp'))
        smtp.sendmail(strFrom, [strTo,strCc], msgRoot.as_string())
        smtp.quit()
    except:
        print "  Error sending mail"
```

Jeremiah Hainly, jhainly@gmail.com

```
                print sys.exc_info()[0]
                sys.exit()


        print "Connected!"
        print "  Sent email to " + targetEmail


################ METHOD 2 ##################
# Send email to user to notify of a reimage #
############################################
def reimage(targetHost, targetFirst, targetEmail, mssNumber, snNumber):
        print "\nConnecting to " + parser.get('email', 'smtp')
        try:
                # Set email from, to, cc
                strFrom = parser.get('email', 'groupEmail')
                strTo = targetEmail
                #strCc = parser.get('email', 'testEmail')   # TEST
                strCc = parser.get('email', 'groupEmail')   # PRODUCTION


                # Create the root message and fill in the from, to, and subject headers
                msgRoot = MIMEMultipart('related')
                msgRoot['Subject'] = 'Workstation Quarantined: '+ targetHost
                msgRoot['From'] = strFrom
                msgRoot['To'] = strTo
                msgRoot['Cc'] = strCc
                msgRoot.preamble = 'This is a multi-part message in MIME format.'


                # Encapsulate the plain and HTML versions of the message body in an
                # 'alternative' part, so message agents can decide which they want to
display.
                msgAlternative = MIMEMultipart('alternative')
                msgRoot.attach(msgAlternative)


                # If the script can't send the HTML, it will send this
                msgText = MIMEText('This is the alternative plain text message. Error
with HTML version')
                msgAlternative.attach(msgText)
```

Jeremiah Hainly, jhainly@gmail.com

```
                # Email text
                # Reference the image in the IMG SRC attribute by the ID we give it
below
                msgText = MIMEText("""\
                        <!doctype html5>
                        <html>
                        <body>
                                <p>Hi """ + targetFirst + """,</p>
                                <p>""" + parser.get('email','teamName') + """ has
identified your workstation with the hostname, """ + targetHost + """, as an infected
workstation. A ticket (INC"""+snNumber+""") has been generated to your local client
support team to reimage the infected workstation. The workstation will be quarantined
from the Hershey internal network within the next hour and will not be allowed back on
the network until the system has been verified as reimaged.</p>
                                <p>Please review the security awareness training available
at """ + parser.get('email', 'trainingSite') + """. Thank you for your cooperation and
understanding as we work to keep yours and the company's information private and
secure.</p>
                                <p>If you have any questions, please reach out to """ +
parser.get('email', 'groupEmail') + """. Please reference incident number """ +
mssNumber + """.</p>
                                <p>Thank you,</p>
                                <img src="cid:image1">
                        </body>
                        </html>
                        """, 'html')
                msgAlternative.attach(msgText)


                # This example assumes the image is in the current directory
                fp = open(resource_path(parser.get('email', 'teamLogo')), 'rb')
                msgImage = MIMEImage(fp.read())
                fp.close()


                # Define the image's ID as referenced above
                msgImage.add_header('Content-ID', '<image1>')
                msgRoot.attach(msgImage)


                # Send the email (assumes SMTP authentication is not required)
```

```
            import smtplib
            smtp = smtplib.SMTP()
            smtp.connect(parser.get('email','smtp'))
            smtp.sendmail(strFrom, [strTo,strCc], msgRoot.as_string())
            smtp.quit()
        except:
            print "  Error sending mail"
            print sys.exc_info()[0]
            sys.exit()


        print "Connected!"
        print "  Sent email to " + targetEmail

#sendmail(hostname, firstName, userEmail, mssTicket, snTicket)
```

sepReimage.py

```
# Purpose: Quarantine or remove quarantine from Hershey endpoint. This is part of a
larger script
#          that verifies the user's identity in Splunk and sends the user an email
#                to notify of a reimage ticket being created.


import warnings
warnings.filterwarnings("ignore")


import requests
import requests.auth
import sys
import os
import ConfigParser


#################### RESOURCE PATH ######################
# Changes the resource path so that the image in the #####
# email can be referenced when compiled with pyinstaller #
# but also referenced when running in Python #############
#########################################################
def resource_path(relative_path):
    # Get absolute path to resource, works for dev and for PyInstaller
    try:
              # PyInstaller creates a temp folder and stores path in _MEIPASS
              base_path = sys._MEIPASS
    except Exception:
              base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)


parser = ConfigParser.RawConfigParser()
parser.read(resource_path('configs.ini'))


########## METHOD 1 ##############
# Submit host to SEP for Reimage #
##################################
def reimage(targetHost, targetCommand):
```

Jeremiah Hainly, jhainly@gmail.com

```
        print "\nConnecting to SEPM"
        # Connect to SEPM's web app port using quarantine user and refresh the access
token.
        try:
                # Client ID = User Name. Client Secret = Password. Account was created
via https://vmsepp01:8446/
                # Refresh token instructions at bottom of code
                response = requests.post(parser.get('sep', 'authurl'), verify=False)
#remote auth=client_auth
                data = response.json()
                access_token = data['value']
                headers = {"Authorization": "bearer " + access_token}
                response = requests.get(parser.get('sep', 'wsdl'), headers=headers,
verify=False)
        except:
                print "Error authenticating to SEPM server. Please verify client_id,
client_secret, and refresh_token"
                sys.exit()
        print "Connected!"


        # Request the SEP GUID for a computer by passing through computer's host name
via SOAP call
        try:
                headers = {"Authorization": "bearer " + access_token, 'content-type':
'text/xml', "SOAPAction":
"http://client.webservice.sepm.symantec.com/getComputersByHostName"}
                body = """
                <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://client.webservice.sepm.symantec.com/">
                        <soapenv:Header/>
                        <soapenv:Body>
                                <ns:getComputersByHostName>

        <computerHostNames>"""+targetHost+"""</computerHostNames>
                                </ns:getComputersByHostName>
                        </soapenv:Body>
                </soapenv:Envelope>
                """
```

Jeremiah Hainly, jhainly@gmail.com

```
                # HTTP POST command. Sends the SOAP commands above
                response = requests.post(parser.get('sep',
'wsdl'),data=body,headers=headers, verify=False)


                import xmltodict
                # Writes the response to the HTTP POST to a dictionary for parsing
                doc = xmltodict.parse(response.content)
                # Access the dictionary and pull the GUIDE
                targetGUID =
doc['S:Envelope']['S:Body']['ns2:getComputersByHostNameResponse']['ns2:ComputerResult']
['computers']['computerId']
                # Print the GUID
                print "   " + targetHost + " SEP GUID: " + targetGUID
        except:
                # Warn the user that other tasks within the script may have been run.
                print "Unable to retrieve host GUID. Please validate hostname and be
aware that I might have completed some tasks already."
                sys.exit()


        # Request quarantine / undo by passing the computer's GUID (found above) via
SOAP call
        try:
                headers = {"Authorization": "bearer " + access_token, 'content-type':
'text/xml', "SOAPAction":
"http://command.client.webservice.sepm.symantec.com/runClientCommandQuarantine"}


                body = """
                <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://command.client.webservice.sepm.symantec.com/">
                        <soapenv:Header/>
                        <soapenv:Body>
                                <ns:runClientCommandQuarantine>
                                        <command>

        <commandType>"""+targetCommand+"""</commandType>

        <targetObjectType>COMPUTER</targetObjectType>
```

Jeremiah Hainly, jhainly@gmail.com

```
        <targetObjectIds>"""+str(targetGUID)+"""</targetObjectIds>
                                        </command>
                                </ns:runClientCommandQuarantine>
                        </soapenv:Body>
                </soapenv:Envelope>"""


                response = requests.post(parser.get('sep',
'wsdl'),data=body,headers=headers, verify=False)
                if targetCommand == "Quarantine":
                        print "  Successfully quarantined " + targetHost
                if targetCommand == "Undo":
                        print "  Successfully removed quarantine from " + targetHost
        except:
                print "Unable to quarantine. Dunno why I failed"
                sys.exit()
```

Jeremiah Hainly, jhainly@gmail.com

servicenowReimage.py

```
# Purpose: Open ServiceNow ticket to reimage user. This is part of a larger script
#          that verifies the user's identity in Splunk and sends the user an email
#                to notify of a reimage ticket being created.


###############################################################################################
#####
#      PLEASE VERIFY THAT THIS SCRIPT IS RUNNING IN TEST BEFORE DOING ANY TEST RUNS
          #
###############################################################################################
#####
######## Imports ##########
# Import required modules #
###########################
import requests # Python -m pip install requests
import re
import sys
import os
import ConfigParser


#################### RESOURCE PATH ######################
# Changes the resource path so that the image in the #####
# email can be referenced when compiled with pyinstaller #
# but also referenced when running in Python #############
#########################################################
def resource_path(relative_path):
    # Get absolute path to resource, works for dev and for PyInstaller
    try:
              # PyInstaller creates a temp folder and stores path in _MEIPASS
              base_path = sys._MEIPASS
    except Exception:
              base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)


# Define parser for configuration file
parser = ConfigParser.RawConfigParser()
parser.read(resource_path('configs.ini'))
```

Jeremiah Hainly, jhainly@gmail.com

```
################### Define Variables ##############################
# Define ServiceNow URL, Hershey Proxy, and ServiceNow credentials
###################################################################
proxies =
{parser.get('production_proxy','proxytype'):parser.get('production_proxy','proxyurl')}
headers = {"Content-Type":"application/json","Accept":"application/json"}


url = parser.get('production_servicenow', 'url')                # PRODUCTION
ServiceNow URL
user = parser.get('production_servicenow', 'user')              # PRODUCTION
ServiceNow Username
pwd = parser.get('production_servicenow', 'pwd')                # PRODUCTION
ServiceNow Password
'''
url = parser.get('development_servicenow', 'url')               # TEST ServiceNow URL
user = parser.get('development_servicenow', 'user')                     # TEST
ServiceNow Username
pwd = parser.get('development_servicenow', 'pwd')               # TEST ServiceNow
Password
'''


############# METHOD 1 ####################
# Method to submit new ServiceNow tickets #
###########################################
def submit(targetHost, targetUser):
        # Try connecting to ServiceNow and submitting a ticket
        print "\nConnecting to ServiceNow"
        try:
                # variable "response" equal to output from HTTP POST via requests
method. Use previously defined URL, auth, proxy, headers
                # Data provides information for each field in the ServiceNow ticket
                response = requests.post(url, auth=(user, pwd), proxies=proxies,
headers=headers ,data='{"impact":"' + parser.get('snOptions','impact') +
'","urgency":"' + parser.get('snOptions','urgency') + '","priority":"' +
parser.get('snOptions','priority') + '","assignment_group":"' +
parser.get('snOptions','assignment_group') + '","short_description":"Reimage
Workstation: '+ str(targetHost)
+'","caller_id":"'+str(targetUser)+'","contact_type":"System","incident_state":"' +
parser.get('snOptions','incident_state') + '","state":"' +
```

Jeremiah Hainly, jhainly@gmail.com

```
parser.get('snOptions','state') + '","category":"' + parser.get('snOptions','category')
+ '","subcategory":"' + parser.get('snOptions','subcategory') + '","comments":"' +
parser.get('snOptions','teamName') + ' has identified the workstation with hostname
['+targetHost+'] as an infected workstation. Please backup the user\'s files to
OneDrive and reimage the machine as soon as possible to prevent further infection on
the network. After reimage, please reset the user\'s domain password."}')
        except Exception as e:
                print(e)
                # If requests method is unable to connect (Wrong password, wrong URL,
wrong proxy, etc.), provide the inputs and stop the script
                print "Failed to connect to ServiceNow. Please make sure the instance is
available."
                print "CONNECTION DETAILS"
                print "  ServiceNow URL: " + url
                print "  ServiceNow User: " + user
                print "  ServiceNow Password: " + pwd
                print "  Hostname: " + targetHost
                sys.exit()


        # Check for HTTP codes other than 201 (Created)
        if response.status_code != 201:
                print('Status:', response.status_code, 'Headers:', response.headers,
'Error Response:',response.json())
                exit()
        else:
                # If requests method is successful, communicate it
                print "Connected!"


        # Decode the JSON response from requests method to return Incident Number
        try:
                snOutput = str(response.json())
                # Regex search the JSON response for the text "INC" and provide the text
until the next non-letter character
                getSnTicket = re.search('(?<=u\WINC)\w+', snOutput)
                # regex search stores the results as a group. Let's put that incident
number into a variable
                snTicket = str(getSnTicket.group(0))
                # Positive feedback
                print "  Successfully created INC" + snTicket + " to reimage " +
targetHost + "\n"
```

Jeremiah Hainly, jhainly@gmail.com

```
        except:
                # Negative feedback if the regex fails. Stops the script
                print "\nCannot find Incident Number"
                sys.exit()


        # Give the new ServiceNow Ticket number back to be used elsewhere
        return snTicket


###################### METHOD 2 ######################
# Method to request the sysID for ServiceNow tickets #
# Used when an update to a ticket needs to be made ###
# since ServiceNow only communicates in sysID's ######
######################################################
def request(snTicket):
        # Change the URL so that it queries for the ServiceNow ticket number
        global url
        tmpurl = url + '?sysparm_query=number=INC' + snTicket #TEST SERVICENOW

        # Try connecting to ServiceNow and getting ticket info
        print "\nConnecting to ServiceNow"
        try:
                # variable "response" equal to output from HTTP GET via requests method.
Use previously defined URL, auth, proxy, headers
                response = requests.get(tmpurl, auth=(user, pwd), proxies=proxies,
headers=headers)
        except:
                # If requests method is unable to connect (Wrong password, wrong URL,
wrong proxy, etc.), provide the inputs and stop the script
                print "Failed to connect to ServiceNow. Please make sure the instance is
available."
                print "CONNECTION DETAILS"
                print "  ServiceNow URL: " + tmpurl
                print "  ServiceNow User: " + user
                print "  ServiceNow Password: " + pwd
                sys.exit()


        # Check for HTTP codes other than 200 (OK)
        if response.status_code != 200:
                print('Status:', response.status_code, 'Headers:', response.headers,
```

```
'Error Response:',response.json())
                exit()
        else:
                # If requests method is successful, communicate it
                print "Connected!"


        # Decode the JSON response from requests method to return Incident Number
        try:
                snOutput = str(response.json())
                # Regex search the JSON response for the text "sys_id" and a few non-
letter chars, then provide the text until the next non-letter character
                getSysID = re.search('(?<=u\Wsys_id\W\W\W\w\W)\w+', snOutput)
                # regex search stores the results as a group. Let's put that incident
number into a variable
                sysID = str(getSysID.group(0))
                # Positive feedback
                print "  Successfully found sys_id: " + sysID + " for INC" + snTicket +
"\n"
        except:
                # Negative feedback if the regex fails. Stops the script
                print "\nCould not find anything"
                sys.exit()


        # Give the sysID back to be used for the update method
        return sysID




################## METHOD 3 ##################
# Method to update old ServiceNow tickets ####
# Used when an incident exists for the issue #
##############################################
def update(sysID, targetHost, targetUser):
        # Change the URL so that it points at the sysID of the ServiceNow incident
identified in the "request" method
        tmpurl = url + '/' + sysID

        # Try connecting to ServiceNow and updating a ticket
        print "\nConnecting to ServiceNow"
```

Jeremiah Hainly, jhainly@gmail.com

```
	try:
		# variable "response" equal to output from HTTP PUT via requests method. Use previously defined URL, auth, proxy, headers
		response = requests.put(tmpurl, auth=(user, pwd), proxies=proxies, headers=headers
,data='{"impact":"1","urgency":"2","priority":"2","assignment_group":"HCOD","short_description":"Reimage Workstation: '+ str(targetHost)
+'","caller_id":"'+str(targetUser)+'","contact_type":"System","incident_state":"-1","category":"PC Software","subcategory":"Antivirus","comments":"Hershey Cyber Defense has identified the workstation with hostname ['+targetHost+'] as an infected workstation. Please backup the user\'s files to OneDrive and reimage the machine as soon as possible to prevent further infection on the network. After reimage, please reset the user\'s domain password."}')
	except:
		# If requests method is unable to connect (Wrong password, wrong URL, wrong proxy, etc.), provide the inputs and stop the script
		print "Failed to connect to ServiceNow. Please make sure the instance is available."
		print "CONNECTION DETAILS"
		print "  ServiceNow URL: " + tmpurl
		print "  ServiceNow User: " + user
		print "  ServiceNow Password: " + pwd
		sys.exit()


	# Check for HTTP codes other than 200 (OK)
	if response.status_code != 200:
		print('Status:', response.status_code, 'Headers:', response.headers, 'Error Response:',response.json())
		exit()
	else:
		# If requests method is successful, communicate it
		print "Connected!"
		print "  Successfully updated incident!"
```

splunkReimage.py

```
# Purpose:     Access the Splunk API to retieve user information.
import sys
import os
import re
import splunklib.client as client
import splunklib.results as results
from time import sleep
import ConfigParser


#################### RESOURCE PATH ######################
# Changes the resource path so that the image in the #####
# email can be referenced when compiled with pyinstaller #
# but also referenced when running in Python #############
#########################################################
def resource_path(relative_path):
    # Get absolute path to resource, works for dev and for PyInstaller
    try:
            # PyInstaller creates a temp folder and stores path in _MEIPASS
            base_path = sys._MEIPASS
    except Exception:
            base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)


# Define parser for configuration file
parser = ConfigParser.RawConfigParser()
parser.read(resource_path('configs.ini'))


########################## Method 1 ###################################
# Search splunk for the inputted user and return their name and email #
#######################################################################
def search(reimageUser):
        #Connect to Splunk Server
        print "\nConnecting to Splunk..."
        try:
                # Connect to Splunk
                service = client.connect(
                        # Connection parameters
```

Jeremiah Hainly, jhainly@gmail.com

```
                        host=parser.get('splunk', 'host'),                      #
Splunk search head addres
                        port=parser.getint('splunk', 'port'),
        # Splunk default deployment server port
                        username=parser.get('splunk', 'username'),          # Admin
profile username
                        password=parser.get('splunk', 'password'))          # Admin
profile password
        except Exception as e:
                print str(e)
                print "\nError connecting to Splunk Server. Please check credentials and
URL"
                sys.exit()


        # Positive feedback after connection established
        print "Connected!"


        #Search Splunk for target user, return email address, first and last name
        job = service.jobs.create("| inputlookup "+parser.get('splunk', 'lookupFile')+"
| search identity="+reimageUser+" | table identity email givenName last")


        while not job.is_done():
                # Wait until search is complete to avoid errors on successful queries
                sleep(.2)
        # Get the results of the query and write into an array
        reader = results.ResultsReader(job.results())
        # Take the values from the array and write to userInfo
        for result in reader:
                userInfo = str(result)


        #Validate that Splunk returned valid results
        try:
                # If userInfo has a value, this will return true, otherwise it will drop
into the "except"
                userInfo
        except NameError:
                # If userInfo does not have a value, negative feedback and close the
script
```

Jeremiah Hainly, jhainly@gmail.com

```
            print "\nInvalid User. Check for typos and verify the user is in Active
Directory.\nIf you're doing it right, then the "+parser.get('splunk', 'lookupFile')+"
Splunk lookup is broken."
            sys.exit()


    # Parse out and display the data collected from Splunk
    print "Here's what I found:"


    # Parse and print User ID
    getUserID = re.search('(?<=identity\W\W\W\W)\w+', userInfo)
    userID = str(getUserID.group(0))
    print "  User ID: " + userID


    # Parse and print Email
    getUserEmail = re.search('(?<=email\W\W\W\W)\w+\W+\w+\W+\w+', userInfo)
    userEmail = str(getUserEmail.group(0))
    print "  User Email: " + userEmail


    # Parse and print first name
    getFirstName = re.search('(?<=givenName\W\W\W\W)\w+', userInfo)
    firstName = str(getFirstName.group(0))
    print "  User First Name: " + firstName


    # Parse and print last name
    getLastName = re.search('(?<=last\W\W\W\W)\w+', userInfo)
    lastName = str(getLastName.group(0))
    print "  Last Name: " + lastName


    # Return User ID, email, and name for user in original method parameter
    return userID, userEmail, firstName, lastName
```