Bachelor Degree Project

UNIVERSITY OF SKÖVDE

AUTOMATING THE BASIC CONFIGURATION OF IPMI INTERFACES

To Reduce the Risk of Misconfiguration

Bachelor Degree Project in Network and System Administration
Level 22.5 ECTS
Spring term 2020

Mark Jack Sundahl

Supervisor: Thomas Fischer
Examiner: Jianguo Ding

Abstract

In this report the lack of basic security of IPMI devices when freshly received from the manufacturer will be discussed and analysed. Furthermore, the rest of the report will focus on attempting to find a solution on how to automate configuration of IPMI devices (and iDRACs due to hardware being borrowed from Ericsson Linköping) to avoid misconfigurations and to change the default credentials shipped. Three proposed workflows will be implemented and compared, and a fourth is further proposed in this report. All three workflows prove to work but the first using OME is found being slower, needing an additional license per iDRAC and require manual intervention while the other two workflows works fully automated. Considering how the three first workflows all require *AutoConfig* regardless of the rest of the steps the second workflow is recommended as its way faster than the other two methods. In theory the fourth workflow would be even better as it does not require *AutoConfig* to work nor would it necessarily only work for iDRACs but work for any IPMI device which supports the *REDFISH API*. Lastly getting better basic security should be something that the manufacturers should strive for as it only strengthens their brand and should they do so in the future then this automation may be unnecessary but could be repurposed to change any setting in the IPMI device.

Table of Contents

1 Introduction

Management interfaces such as Dells iDRAC, HP iLO and many more are systems that allows a system administrator to remotely configure and access servers as if they were in front of the server without the need to install any additional software. The servers do not even have to have an OS (*Operating System*) installed at all. These interfaces are based on the IPMI standard (*Intelligent Platform Management Interface*) which specifies management and monitoring of the host machines CPU, firmware (BIOS or UEFI) and OS. Some of these management interfaces allows for even more control such as allowing mounting of network shared media or power management of the physical server (as the interface may be looked at like a separate machine which may be running even though the physical machine is not).

Before IPMI was introduced, system administrators would have to install and configure tools on each server that would be OS dependant and would not function without the server OS running. As such there was no way for a system administrator to install an OS on a server without being physically on site and using the server.

Many of these management interfaces are accessed through a web interface with some sort of authentication. However, unconfigured interfaces are in many cases set with a default account and password which could lead to a security risk if not changed as it would mean unmitigated access to the physical machine through this interface.

The advantages of using management interfaces is, however, rather many as with the access that it gives a system administrator it allows for monitoring of the physical hardware remotely without having to rely on any other external tools. This becomes especially interesting in a homogeneous environment (in this case meaning servers from the same manufacturer) as you can then interface with different servers using the same method regardless of what OS the host machine is running. It can also give access to starting up the server, should it be powered off and power is available. And it only becomes more powerful when combined with scripting as scripts can make use of API (*Application Programming Interface*) calls to configure and change settings using the management interface. These management interfaces are sometimes referred to as out-of-band management while in-band management would be to use SSH or similar tools which are based on software installed on the host OS.

Since there is a lack of security with these base implementations, this report will focus on attempting to find a way to automate the deployment of IPMI configurations, hence a system administrator that has configured such an environment does not need to manually edit the configuration of the IPMI interface, yet still have better security. This does not however, change the recommendation from the manufacturers (however unclear that recommendation may be as some manufacturers do not state this fact) that the IPMI interfaces should not be accessible through the Internet. Automating these settings means less risk of something getting typed wrong in addition to making sure that settings that should be default but is not is indeed set.

In bigger companies it is not unusual for different teams to deal with separate parts of the environment and in such situations it is not feasible to expect that human communication will be sufficient to deal with the numerous different requirements and changes of configurations that may be required for other teams. As such, automation should be used when possible to deal with repeated and mundane tasks. This way the human factor be it in the form of missing a warning or simply not knowing something changed is eliminated at the cost of more knowledge required to implement such automation. This applies to both initial (such as initial configuration of a machine or similar) and tasks that are made in response to some event, as automation will always do the job as described and it will do it faster if it is triggered automatically, compared to a human doing the job manually. In this case talking of the extremes of the spectrums with either full automation or human intervention with no help of scripts. A human with the aid of scripts can accomplish tasks just as fast as full automation if only counting the time to do the actual task and not counting the time it takes to act (Nagy, Christensen and Horne, 2019).

To make matters worse for manual configuration is that humans tend to be inconsistent with large inputs, humans are better suited for designing and making a system once and not for repeating the same task over and over with small differences. As such it is better to automate configurations when possible and let the system administrator design the configurations once and let the system deal with the implementation of said configuration. This may also reduce the risk of certain configuration combinations that reduce security or functionality being applied accidentally (Witte, Cook, Kerr and Shaffer, 2012).

Gregg (2017) states that security is all about reaching a balance between security and usability. He points this out by comparing a system that is completely detached from any network with one that is directly connected to the Internet without any firewall, anti-virus or security patches. The former system would be considerably more secure against intrusion but much less accessible than the latter. In a similar way security in IPMI based interfaces needs to strike a balance between security and usability to be useful for system administrators while still managing to have acceptable security. Furthermore, Gregg (2017) states that the choice is always between loosing security for usability or loosing usability for security. Such a consideration should always be made to strike the best balance for that implementation or feature. This goes hand in hand with the *CIA* (Confidentiality, Integrity and Availability) triad of information security. See figure 1 for the *CIA* triad in illustration (Gregg, 2017).

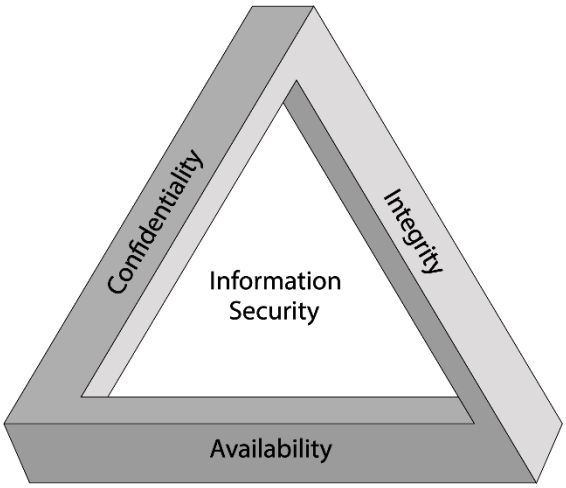

Figure 1: The CIA Triad

Confidentiality addresses secrecy and privacy of information. This means that it is about protecting the data similarly to how locked doors protect physical assets. Integrity implies the correctness of information. This does not mean that the data necessarily is accurate, but it means that it has not been modified in transit or storage. Lastly availability means that when an authorized entity (person or system) needs the information it should be available for the entity (Gregg, 2017).

Another big factor for security implementations is costs, as better security usually cost more to implement be it in forms of educating staff, time, or software/hardware purchases. As such the balance to strike is not only between security and usability but also cost (Modig, 2019).

The rest of this report will attempt to find a solution to automate configuration of IPMI interfaces to have a better basic security configuration. The report will also be split into 5 more sections apart from this. Section two is about previous scientific work and their conclusions. The third section will go over the method that will be used in this report to gain further insight. Section four will summarise the information gained through the study. The fifth section will analyse the results from the previous section, but it will also contain a part about ethics. Finally, the last section is about potential future work that can be done to further investigate or improve on the topic.

2 Background

Hong and Xiongfei (2017) discussed IPMI regarding its security both in specification but also regarding manufacturers implementation of it. In their introduction they quickly summarise that for smaller companies it is common to have system administrators manage servers directly however, for bigger companies this is not feasible and tools like SSH (*Secure Shell*), RDP (*Remote Desktop Protocol*) or even third party tools may be installed to accomplish this task. Though relying on such tools requires the host OS to be running as they are in-band types of management. Using such tools would also increase network traffic load as well as reduce security of server management. Hong and Xiongfei (2017) also note that since its introduction BMC (*Baseboard Management Controller*) has become an equipment standard for network and server hardware because of the features they add. IPMI has some vulnerabilities on its own (in the standard) before manufacturers make their implementation such as servers often having the same IPMI password and that passwords need to be stored in an unencrypted way. To add to this each vendor implements their own solution to IPMI which usually use the least secure options by default for the management interface (Hong and Xiongfei, 2017).

Bonoski, Bielawski and Halderman (2013), wrote about IPMI interfaces and their issues where they found that not only is the specifications very lacking regarding security but also in implementation as they found several exploits in SuperMicros' implementation of IPMI. These exploits include no server-side validation of input, buffer overflow and shell injection. They also noted that they managed to find 105 334 IPMI interfaces connected to the Internet in 2013 even though this goes against all recommendations of the standard, and this number was probably on the lower side on how many servers that were connected wrongfully. Out of those 105 334 machines 41 545 was susceptible to the different exploits they discovered. They also noted that in many cases the manufacturer also add default accounts for the IPMI interface that if not configured could be exploited to gain access, this without even using the buffer overflow attack they found that can be used on the login page. In their paper they also discuss the different attack scenarios that could be exploited on IPMI interface devices such as subverting the host system, BMC spyware, persistent BMC rootkits, attacks on the BMC from the host and IPMI botnets. Because the BMC usually runs closed source code on top of normal OS, they can be vulnerable to exploits with little to no chances for the administrator of the server to notice something being wrong. They conclude that manufacturers should put more thought into security in order for systems to not be vulnerable by default, and that administrators should ensure that IPMI interfaces are always up-to date, do not use default credentials and are not publicly accessible on the Internet (Bonkoski, Bielawski and Halderman, 2013).

White, Litkey and Calvert (2005) discussed the need for a set standard in autonomous server management systems. They noted that combining monitoring tools and automation of IPMI with other controlling interfaces, could be used to automatically deal with events that would otherwise take a system administrator some time and effort to fix when it could be relatively simple to automate. One such simple example would be to restart a server should it not

respond to heartbeats in time, though more complex issues and solutions could also be managed in such a way. They also noted that these management solutions need to have the option to be hot-swappable as their operation and possible software upgrade should not in any way interrupt the server´s host OS. However, as opposed to the general IPMI specifications their solution includes PAM (*Pluggable Authentication Modules*), firewalls and SSL (*Secure Sockets Layer*) encryption making it more secure. In addition to this they also noted that any such automated management system needs to be on separate hardware from the servers, as to be able to function when the base server is malfunctioning and as to not have the server host OS and management system interfere with each other due to having to compete for resources (White, Litkey and Calvert, 2005).

Leangsuksun et al (2006) did a study where they used and compared IPMI monitoring to other monitoring solutions where they managed to make a more responsive system by polling the data from the IPMI device instead of from the host OS. By doing this, responsiveness and load on the host was reduced and because of those factors it scales better in larger environments over traditional monitoring solutions. In their solution they also implemented an intelligent filtering solution that looks at the history of events that had already happened to attempt to make a more informed decision on the event. This means they can filter out events which may not be errors but simply degradation down to wear and tear of hardware or external factors such as ambient temperature change (Leangsuksun et al, 2006).

Moore (2013) posted an entry about the lack of security with IPMI interfaces as they use cipher 0 in the IPMI 2.0 standard which as a result means that authentication could be bypassed allowing anyone to get access to the BMC and make changes. Furthermore, he also notes that some BMC allows for anonymous authentication by default which also has administrative privileges. Both settings are dangerous as they can be exploited to make any changes to the IPMI including making new administrative users or change the passwords of existing ones (Moore, 2013).

3 Research Approach

The goal of this report is to attempt to improve the security of IPMI interfaces as they are a valuable tool for system administrators, but the unconfigured IPMI interfaces are simply not secure enough for use. To further this goal, this section will contain both the research question that this report attempts to answer but also the implementation of an automated system to attempt to improve the security of the IPMI interfaces.

3.1 Research Question

This report will attempt to focus on answering the question: “Given the documented security issues of IPMI systems, how can automated systems be used to address (or mitigate) those issues?”.

Answering this question would be beneficial to both users and administrators as system administrators can keep using IPMI devices with all their benefits, and in the case of it not getting done manually it would be an improvement for users of such a system as it makes it less likely that the system will get compromised and the users information leaked.

It should be noted however, that even if some problems can be solved with automation, unless the manufacturer implements secure features like encrypting traffic when using IPMI there is little that can be improved from that standpoint as the firmware that the device is using needs to be improved.

3.2 Acknowledgements

Many thanks to Ericsson Linköping for lending me hardware to test these implementations and for the help received. Because the work was done on their machines an NDA was signed in where no internal structure of Ericsson may be revealed through this report. Also, many thanks to Thomas Fischer for supervising me in this task and helping me to polish this report. The NDA will be further discussed in chapter 6.1.

3.3 Proposed Workflows

To test the possibility to make the IPMI devices more secure through automation with these management interfaces, three different workflows (ways to accomplish the task) supplied by Dell will be implemented. The reason for using Dell machines is because Ericsson Linköping were kind enough to lend their hardware for this test in exchange for evaluating these three methods of automation. However, since Dell iDRAC (*Integrated Dell Remote Access Controller*) is based on IPMI it would not be unreasonable to assume that similar implementations could be possible on other manufacturers.

- Auto configuration of baseline and deployment of final profile from HW Management System (*Dell OpenManage Essentials*)

- Auto configuration of final profile which is prepared by a script
- Auto configuration of baseline and deployment of final profile by a script

All the methods do require a DHCP server configured with Dell *AutoConfig*. This is a tool that Dell has made in which the iDRAC gets and IP from a DHCP server, as well as getting information about where a NFS or CIF share is located which holds configuration files for the iDRAC which it can apply automatically.

The first workflow is based on OME (*OpenManage Essentials*). In this workflow the OME service would look for and add the iDRAC to its inventory after it has been configured with a baseline configuration from *AutoConfig*. After that, a manual task is needed to supply OME with the final configuration of the iDRAC which OME will then push to the iDRAC for application. Due to the IP changing on the final configuration application, another manual task is needed to remove the iDRAC from OME and let it be rediscovered later automatically, should management through OME be wanted. On top of this a script would be beneficial to be ran after it has been rediscovered to add the device to a static group as those are easier to work with in OME. See figure 2 for an illustration of the planned workflow.

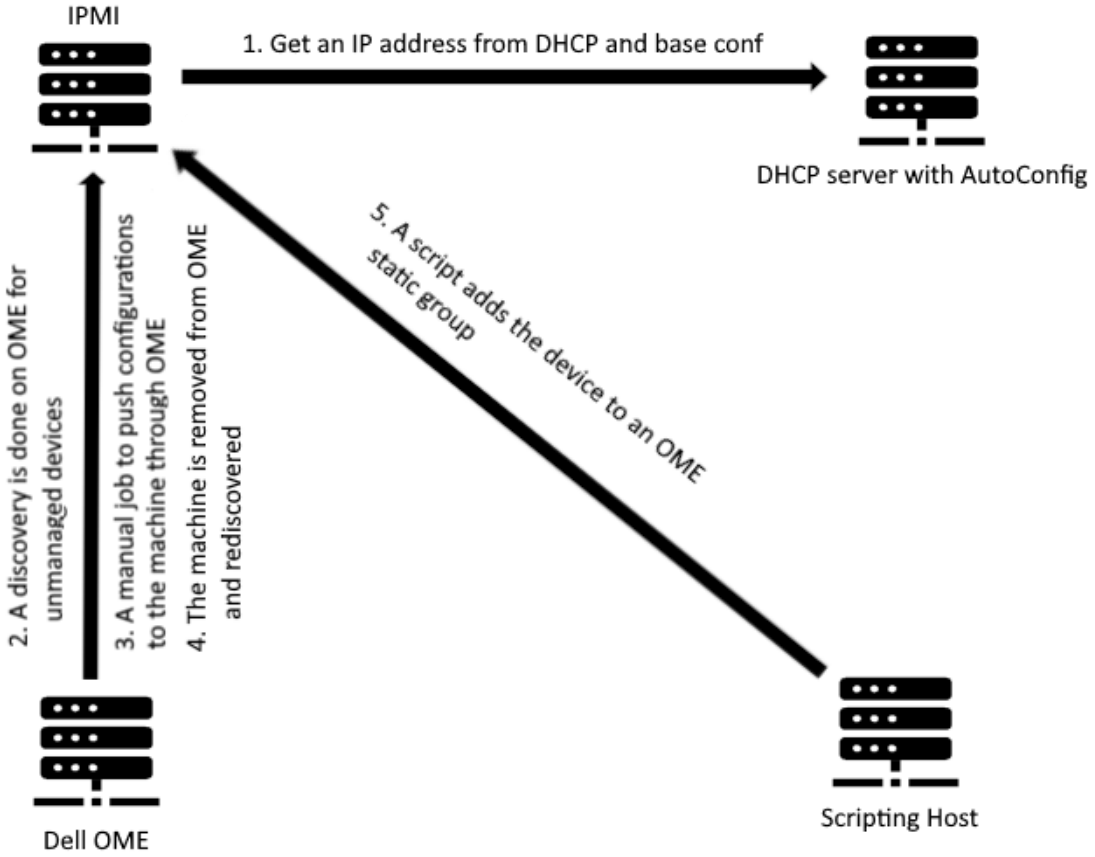

Figure 2: Configuration with AutoConfig and OME

In the second workflow a script should fetch information from a database that contains the relevant information on how the iDRAC should be configured before the server is connected to the network for the first time. This script should then generate a unique file which is put on the network share that Dell *AutoConfig* points to. The configuration would then automatically be applied once the iDRAC connects to *AutoConfig*. This file is the final profile for the iDRAC and will contain the base config and any device specific setting that should be applied to the iDRAC. See figure 3 for the steps taken to accomplish this automation.

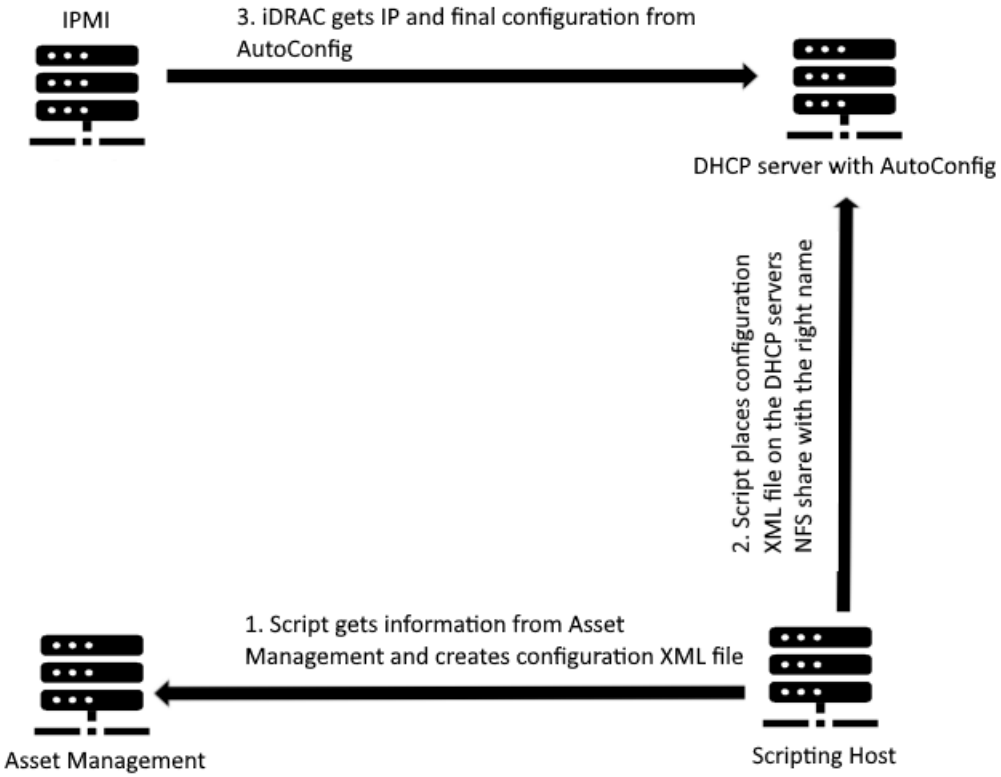

Figure 3: Configuration with AutoConfig and NFS share

In the last workflow the iDRAC will first grab a base config from *AutoConfig* and then once that has been applied and the machine is ready for more settings, it will be detected by a script that generates the device specific settings from the database and then push the final configuration to the iDRAC using the *Redfish API*. See figure 4 for the steps for this automation.

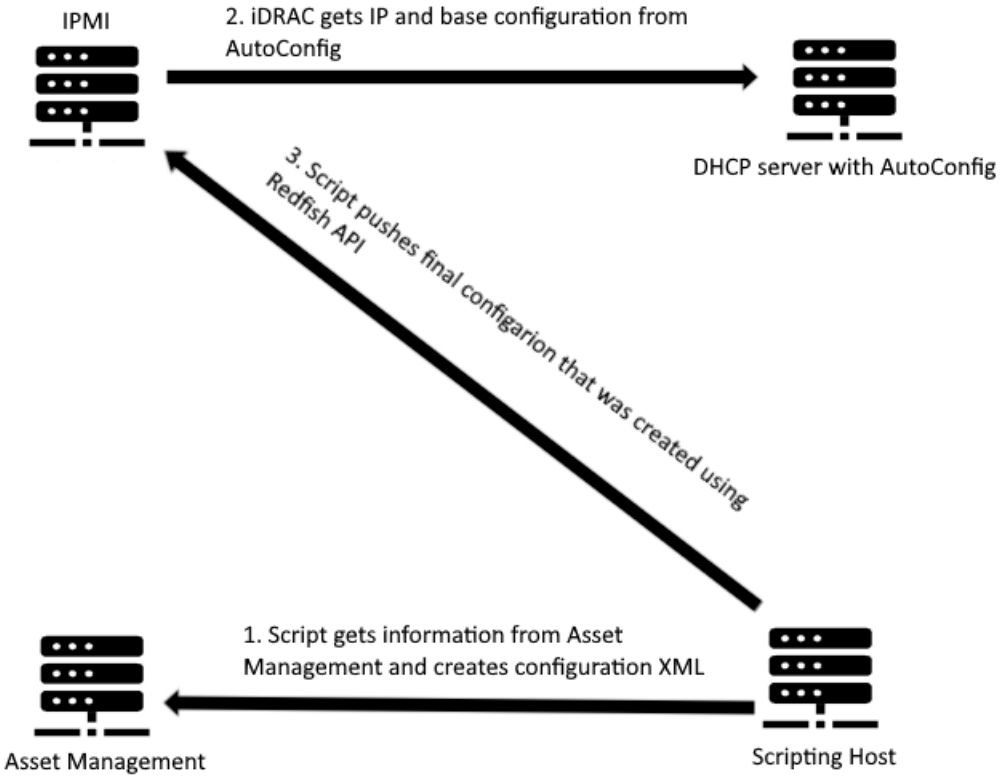

Figure 4: Configuration using AutoConfig and Redfish API

The difference between the second and the last workflow is only that the last workflow does not need to store any files on a share, as it generates the configuration as it is needed but instead needs to be run after the initial configuration using *AutoConfig* has been done.

Once implemented the workflows will be compared in level of automation (that is to say fewer manual steps is better as it lessens the chance for a bad input from a human) as well as performance of the workflows.

4 Implementation

The first thing that needs to be done is to install a machine with an OS to host a DHCP server for *AutoConfig*. In this case a *CentOS* machine was installed for this purpose and the package *dhcp* was installed. This machine was also setup to host a NFS share as that is what the iDRAC´s will pull from when using *AutoConfig* (It is possible to also have it hosted on a CIF share). The required parameters for *AutoConfig* was inserted into the DHCP server's config, which is the option 43, an example of such a configuration can be found in Appendix C.

This example was taken from Dell directly as the NDA prevents disclosure of any IPs, hostnames, or other internal structure of Ericsson which would be otherwise revealed in this configuration. The important parts to note is that option 43 is identified and set. Since the *iDRAC´s* use "iDRAC" as its VendorID which is why it must be specified. This is also where the *iDRAC* will get redirected to the NFS share to grab its initial configuration for application. As long as the iDRAC version is higher than 2.20.20.20 then a filename does not need to be specified as the iDRAC will attempt to first find a file with servicetag (that being the unique identifier of a physical server) in the name, if that's not found then it will look for a file with its model number and lastly if none of those were found it will attempt to look for a basic configuration file with no prefix to the name (Dell, n.d.b.).

AutoConfig as specified above would enable iDRACs to connect to the DHCP server and be redirected to a CIF/NFS share which holds configuration files that the iDRAC will download and apply.

To make a base config that changes the default user credentials is uncomplicated and the same applies to all three methods. It needs to be a correctly formatted xml file and the only fields that are needed are the ones that needs to be changed. As such the only fields that the base configuration in this case has, is the ones about user number 2 (as that is the default admin account).

```
<SystemConfiguration>
<Component FQDD="iDRAC.Embedded.1">
<Attribute Name="Users.2#UserName">test2020</Attribute>
<Attribute Name="Users.2#Privilege">511</Attribute>
<Attribute Name="Users.2#IpmiLanPrivilege">Administrator</Attribute>
<Attribute Name="Users.2#Enable">Enabled</Attribute>
<Attribute Name="Users.2#SolEnable">Enabled</Attribute>
<Attribute Name="Users.2#ProtocolEnable">Disabled</Attribute>
<Attribute Name="Users.2#AuthenticationProtocol">SHA</Attribute>
<Attribute Name="Users.2#PrivacyProtocol">AES</Attribute>
<Attribute
Name="Users.2#SHA256Password">4699F99853C51DA1BA9559D5CAD25C74DB4C4DB4980A1C960CE9C50E
24ADE738</Attribute>
<Attribute Name="Users.2#SHA1v3Key">36ebf9393d33594d99af7548ad8bce8d12a199ea</Attribute>
<Attribute Name="Users.2#MD5v3Key">4df181fb93f8525e4ff973ab590c686f</Attribute>
<Attribute Name="Users.2#SHA256PasswordSalt">C65A6070200CDD6289C9B69ED01DD771</Attribute>
</Component>
</SystemConfiguration>
```

In this example the default root account is replaced with a user called “test2020” with the password of “Syp9393”. Any iDRAC options can be changed in this way. The base config only contains the basic options that all iDRAC in the environment should have, the rest gets specified in the final configuration. The second implementation will not apply a basic configuration as it will grab a final configuration using *AutoConfig* automatically while the other implementations apply the base config first and then in different ways apply a final configuration as mentioned later.

Configurations can be extracted from iDRACs using either the *REDFISH API* to create an XML file from the current settings, they can be extracted through OME or they can also be extracted using *RACADM* (Dell Remote Access Controller Administration). However, this is outside the scope of this report.

The first implementation is realised through a piece of software Dell has made which is called OME. Through this system a discovery task is made to scan for new devices and add them into the OME inventory of machines, with the credentials that was added from the base configuration using *AutoConfig*. A query group (query groups are groups that are dynamic in OME which can contain all machines starting with a specific IP as an example) is also made to help with adding the newly found machines into a static group which is also made. Lastly a configuration template is made that contains the relevant settings of the iDRACs. This method will require manual intervention once the machine has been added into OMEs inventory, to deploy the configuration template to the machine as that is not something currently possible to schedule in OME. Once the deployment is complete the machine needs to be manually removed from OME if the IP of the device was changed as OME does not update this on its own. Once the IP has been updated in OME through another discovery, the machine can be added into the static group by a script. The script can be found in appendix A. See the figure 3 in chapter 3.3 for the steps taken to accomplish this automation.

In the second implementation a script was made that fetches information from a database that contains the relevant information on how the IDRAC should be configured before the server is connected to the network for the first time. This script then generates a unique file which is put on the network share that Dell *AutoConfig* points to. The configuration will then automatically be pushed to the machine once it connects to *AutoConfig*. This file is the final profile for the iDRAC and will contain the base config and any device specific setting that should be applied to the iDRAC. The script can be found in Appendix B. See figure 4 in chapter 3.3 for the steps taken to accomplish this automation.

In the last implementation the iDRAC will first grab a base config from *Dell AutoConfig* and then once that has been applied and the machine is ready for more settings, it will be detected by a script that will generate the device specific settings from the database and then push the final configuration to the iDRAC using the *Redfish API*. The script can be found in Appendix

B, note that it is the same script as in the second implementation but that a flag is set in the start of the script, if it is supposed to be implementation 2 or 3 (as the first does not involve this script). See figure 5 in chapter 3.3 for the steps for this automation.

Dell has published python and powershell scripts that allows manipulation of iDRACs through *REDFISH API* which they host on Github. The last implementation uses one of those scripts to push the XML configuration file to the iDRAC once they have been generated. See appendix D for the script made by Dell (Roemer, 2020).

5 Results

Dell has an option now to not ship iDRAC with the default root user credentials, but it is not default, it is a free of charge option for iDRAC (Dell, n.d.d.).

For *AutoConfig* to work with a final profile as in the second implementation, the iDRAC needs to be firmware version 2.20.20.20 or greater as otherwise a specific filename needs to be specified in the DHCP server configuration, making a final profile for each server impossible. Another drawback with *AutoConfig* is that it needs to be in the same subnet as the NFS/CIF share server. This could be worked around by having the *AutoConfig* network only as a staging network for the iDRACs and then in the final configuration move the iDRAC to another subnet where it is intended to be (Dell, n.d.a.).

To be able to work with the first implementation the iDRACs needs a specific license or they will not be added to OME. This means that this implementation is more costly than the others, in addition to not being able to be as automated as the others and with the limitations of clients as specified below (Dell, n.d.c).

Dell OME has another drawback in the form that it can only support up to 8000 devices per *OME* instance. In cases where the server farm exceeds this number, that implementation is suboptimal at best because of this limitation (Dell, 2019).

The third and last implementation requires *Redfish API* support which in turn is only available on iDRAC 7 firmware version 2.40.40.40 and later. As such it cannot be integrated into environments where the iDRACs are of a lower version then that (Dell, n.d.e).

The first implementation also have performance issues as OME does a lot more than what is needed, to solve this particular task and it also involves manual tasks, both for making a final profile to push to a device but also to remove the device once the profile has been deployed.

The second implementation is the fastest by a slim margin, as the script does not need to look at what IP addresses are used currently and match with the right iDRAC, and the iDRAC only have to apply a setting once as it gets the final profile directly as opposed to the third implementation, where it first applies a basic config from *AutoConfig* and then gets a final profile pushed from the script.

Considering these drawbacks and limitations, the second implementation would be preferable as it does not require any additional licenses and an NFS/CIF share is needed regardless of implementation as all three build on *Dell AutoConfig*. This would mean that in the event that a new server is purchased, an entry should be made into the database about the new server with the additional iDRAC information. And the script would then prepare the final configuration before the machine arrives, ready to be used by *Dell AutoConfig*. It should be noted that the second implementation is not fully automated either as by default iDRACs are not configured to use the network properly by asking for a DHCP lease however, the

administrator responsible for plugging in the server only have to go into the iDRAC and enable DHCP and *AutoConfig* and the automation should take over from there instead of having to supply all information manually, risking entering inaccurate information.

OME Discovery Time	**OME Push Times**	**NFS AutoConfig Times**	**Direct Push Time**
3:17	0:41	0:09.084279	1:38.451352
3:25	0:37	0:05.662361	1:36.364070
3:24	0:55	0:03.821112	1:42.028054
3:34	*Fail*	0:04.694050	1:36.950972
3:22	0:57	0:03.795095	1:36.194820
3:25	0:46	0:04.801468	1:35.648945
3:23	1:31	0:04.379154	1:35.642729
3:33	1:19	0:05.040183	1:41.211671
3:32	0:39	0:06.246946	1:37.098824
3:30	0:39	0:05.695747	1:38.609861

Table 1: Benchmarks of the Implementations

The times are in the form of min:sec:msec. The times for OME does not include msec as OME only reports in seconds as the smallest value, Also, note that the NFS times does not include the time for the iDRAC to apply the configurations as no method of timing that was discovered. Although both OME and direct push methods require a basic configuration push of AutoConfig which is not measured either therefore the time it takes for NFS method to apply should be negligible compared to the others as they have the same method of configuration not counted in the times as well. The times noted in for NFS AutoConfig and Direct push are taken from the python scripts awareness in time as can be observed in appendix B and as such may not be factual down to the last fractions of a second.

Regarding performance the first implementation utilising OME took in total (both discovery and pushing of configuration) an average of 4 minutes and 19.4 seconds (breaking it down for discovery and push give an average of 3 minutes 26.5 seconds and 53.77 seconds respectively) however, OME had a failure during one of the tests. This fail has not been taken into the calculation of average time but it should be noted as it does means this method is less stable and it also requires manual intervention, as previously mentioned which also has not been added to the time in the results. OME can only schedule a discovery to be done once a

day as the most frequent option. It can be forced to run manually but if the idea is to attempt to automate things this would become a hindrance, especially since in the workflow the device has to be removed from OME after push of configuration to then be re-added later with the new IP.

The second implementation using the NFS share had an average of 5.3217 seconds to generate the configuration file and place it onto the NFS server. This method had no failures and no manual intervention needed.

And the last implementation that pushes the configuration directly had an average of 1 minute and 37.8195 seconds to complete a full generation of configuration and pushing to the iDRAC. This implementation also had no failures and no manual intervention needed.

The sample data is rather small and no real conclusion can be taken from it, with such a small sample size other than that it is a big difference between the implementations in regards to the time it takes for the tasks to complete. The reasoning why the second implementation is that much faster than the last is most likely down to not having to probe the network for where each iDRAC device is.

To summarise the limitations and drawbacks of the different implementations mentioned in this report look at the following lists.

Implementation 1 (OME):

- Each server needs an additional license to be able to work with OME
- For *AutoConfig* to work as specified in this report the NFS/CIF share must be on the same subnet as the DHCP server
- For *AutoConfig* to work with no filename specified in the DHCP options the iDRAC needs to be on firmware 2.20.20.20 or greater
- OME can support a maximum of 8000 devices per OME instance
- Slowest implementation
- Manual steps involved

Implementation 2 (final configuration on the NFS share):

- For *AutoConfig* to work as specified in this report the NFS/CIF share must be on the same subnet as the DHCP server
- For *AutoConfig* to work with no filename specified in the DHCP options the iDRAC needs to be on firmware 2.20.20.20 or greater
- Fastest implementation

Implementation 3 (Redfish API):

- For *AutoConfig* to work as specified in this report the NFS/CIF share must be on the same subnet as the DHCP server
- For *AutoConfig* to work with no filename specified in the DHCP options the iDRAC needs to be on firmware 2.20.20.20 or greater
- iDRAC need to be on firmware 2.40.40.40 or greater to have Redfish API support

Any of the proposed solutions could also be used to change settings after the fact by either redoing the deployment task in OME or indicate for the script that the server needs a new configuration. Although the first two implementations are less adaptable in this way as OME needs another manual task and the second implementation requires a manual task of going into the iDRAC settings and enable DHCP and *AutoConfig* once more, this for each server. The third and potentially fourth (fourth being mentioned in future work) implementations and workflows should work without the need to change anything in the iDRACs or any other manual task then to indicate for the script that it needs to be redone as both of these implementations push the configuration directly to the iDRAC with no need for *AutoConfig* at that stage.

6 Conclusions

This report has focused solely on Dell´s implementation of IPMI because of the accessibility to Dell servers through Ericsson Linköping. It would be beneficial for the community at large if there was an automated implementation that worked across all manufacturers. This, however, would mean that the manufacturers would have to come together and make it into a standard, but the advantages of such a solution should outweigh any potential sales pitch of having a specific implementation. This could potentially in the future be added into the IPMI standard, thus forcing the manufacturers to add it into their implementations. It should be noted that *Redfish* API is supported on more IPMI devices as a result, it could be considered a standard of sorts for these types of automation, but it would be better to have it straight into the IPMI standard as to force manufacturers to comply. Dell now supply an option to have the user credentials for the root account to be random, it should be the default shipping options of servers in the case that a system administrator would forget to change the defaults, similarly to how consumer routers when bought from a retail store have the credentials random and printed on the bottom of the device. This is something that other manufacturers should do as well.

When it comes to time potentially saved by automating these steps, it would vary greatly on how much that would have to be set on the iDRAC (such as NTP, LDAP integration, account details and much more) making such a comparison hard to make. Although the fourth implementation (more details in future work) also skips the process of logging into the iDRAC to enable DHCP and *AutoConfig* manually as well as any other settings. Though considering all but the fourth implementation requires enabling of DHCP and *AutoConfig* manually that time could be neglected. At that point, the comparison would be simply of setting the settings correctly which would depend greatly on how much needs to be set. Though the biggest gain here is not time taken but the fact that misconfigurations does not happen in implementation (it could potentially still happen when adding data into the database, which could be alleviated somewhat by having references of weak entities) and would be a matter of set and forget.

No device should be left with known credentials active on the machine, especially not something like an IPMI device which could be used to gain access to the rest of the physical server. As such, changing the default credentials should be the absolute minimum that is done to secure a device. The automation attempts as shown in this report will be able to accomplish this but could also be used to change any settings of the IPMI device. One such example could be to change the default credentials to something only a few people would have access to and then enable LDAP integration to let anyone who needs access use their normal LDAP account for access into the IPMI device.

This way of automating the settings of an IPMI device would be beneficial for any organisation which continually change their environments or install new servers. It would not be too beneficial for an individual on their own home server as it is some effort to set up this

kind of automation. For example, a hosting company that has a lot of servers in a server farm could implement one of these proposed workflows to help automate the settings of these IPMI device.

The IPMI standard should also be updated to encrypt traffic and the user database by default as that is a potential security risk.

6.2 Ethics

Because the work was done with Ericsson a NDA was signed in where no internal component or workings of Ericsson can be revealed. As such this report will not contain any specifics regarding IPs, hostnames or similar information. In such cases where such information is shown it has been replaced with a random value instead. This should be noted when reading and analysing this report, but it should have no considerable impact on the results provided.

6.3 Limitations of Scope

Because the study was only done on Dell servers this report can only make assumptions on other manufacturers implementations of IPMI interfaces. This means that this report heavily use Dell tools such as *AutoConfig* to achieve its results, though a quick search does seem to support the notion that HP implementation *iLO* also supports the *Redfish API* and has the ability to request IP addresses from a DHCP server. Support for *Redfish API* as well as DHCP support would be sufficient to implement something like the fourth implementation (further touched on in future work) of this report on such devices.

7 Future Work

To continue this work another study could be made on another implementation of IPMI such as HPs iLO, or attempt to look at them all.

Another way to progress would be to attempt to look at the IPMI interface itself and see if there is any way to increase the security directly on the IPMI interfaces however, this would probably require a lot of collaboration with a manufacturer. If such a thing could be accomplished then it would need to be added to all the devices out of the box from the manufacturer to ensure the security being applied and not be an optional setting that could be missed.

A less technical study could also be made to look at the knowledge of the security risks amongst system administrators using machines with IPMI interfaces. This could be interesting to see as the problem could be shown to be more or less serious depending on the outcome of the study. This both regarding internet connectivity of the IPMI devices but also to a less extent the default user credentials of the interface.

Lastly another implementation could be attempted where *AutoConfig* is not used at all but otherwise attempt to push configurations directly to the IPMI device through the *Redfish API* before the device is even configured for network use. This could be realised by plugging into the IPMI device directly and use the default IP of the IPMI to push configuration to. With minor adjustments to the settings this should work with any IPMI that has *Redfish API* implementations and as such should not be limited to only Dell. This last workflow could be illustrated with figure 5 and the script in appendix B also has this functionality but could not be tested as physical access is needed and could not be done in this study.

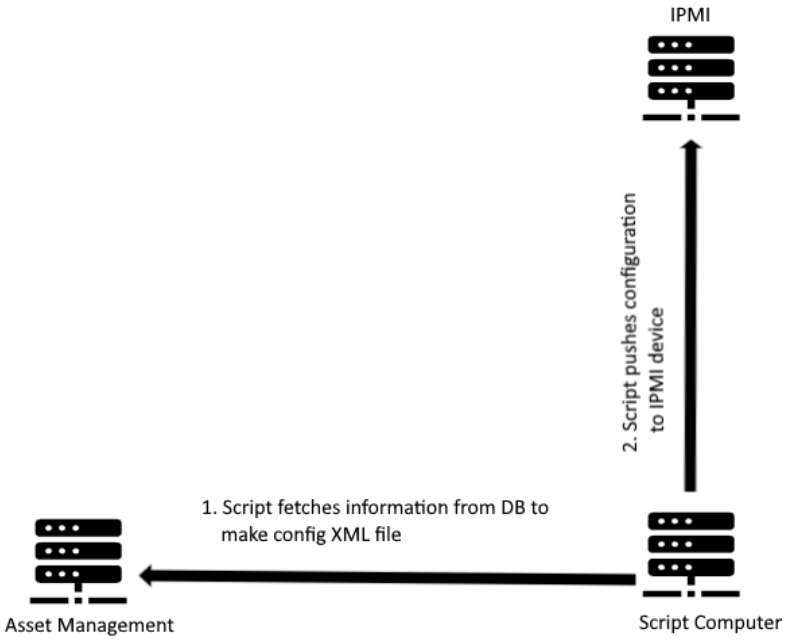

Figure 5: Direct push through Redfish without AutoConfig

8 References

Bonkoski, A., Bielawski, R. & Halderman, A. (2013). Illuminating the Security Issues Surrounding Lights-Out Server Management. In *7th USENIX Workshop on Offensive Technologies*, Washington D.C., August 13, 2013: USENIX Association. https://www.usenix.org/system/files/conference/woot13/woot13-bonkoski_0.pdf

Dell. (2019). *Dell EMC Openmanage Essentials Version 2.5 Release Notes*. 1st ed. https://topics-cdn.dell.com/pdf/openmanage-essentials-v25_release-notes_en-us.pdf [2020-04-16]

Dell. (n.d.a). *Idrac 8/7 V2.30.30.30 User'S Guide*. https://www.dell.com/support/manuals/se/sv/sebsdt1/poweredge-r730/idrac8_2.30.30.30_ug/configuring-servers-and-server-components-using-auto-config?guid=guid-2fdb4eb1-c4bc-481d-8dd7-0240472ca961&lang=en-us [2020-04-16]

Dell. (n.d.b). *Integrated Dell Remote Access Controller 8 (IDRAC8) And IDRAC7 V2.20.20.20 User's Guide*. https://www.dell.com/support/manuals/se/sv/sebsdt1/iDRAC7-8-with-lc-v2.20.20.20/iDRAC8_2.10.10.10_ug-v2/configuring-option-43-and-option-60-on-linux?guid=guid-8ce99f72-e3ae-47aa-8f7f-72e218fdb447&lang=en-us [2020-03-25]

Dell. (n.d.c). *Openmanage Enterprise Licensing Guide*. https://www.dellemc.com/en-nz/collaterals/unauth/offering-overview-documents/products/servers/openmanage_enterprise_licensing_guide.pdf [2020-04-27].

Dell. (n.d.d). *Rackservern Poweredge R340 1U För Små Företag | Dell Sverige*. https://www.dell.com/sv-se/work/shop/servrar-lagring-och-n%C3%A4tverk/smart-value-poweredge-r340-server-basic/spd/poweredge-r340/per3401b [2020-04-15]

Dell. (n.d.e). *Redfish API With Dell Integrated Remote Access Controller*. https://www.dell.com/support/article/sv-se/sln310624/redfish-api-with-dell-integrated-remote-access-controller?lang=en [2020-04-16]

Gregg, M. (2017). *Certified Ethical Hacker (CEH) Version 9 Cert Guide, Second Edition*. 2. ed., Indianapolis: Pearson.

Hong, X. & Xiongfei, Z. (2017). Analysis of Out-of-Band Management Security Based on IPMI Protocol. In *2017 Global Conference on Mechanics and Civil Engineering (GCMCE 2017)*. Guangzhou: Atlantis Press, pp. 102-106. doi: https://doi.org/10.2991/gcmce-17.2017.20

Leangsuksun, C., Rao, T., Tikotekar, A., Scott, S., Libby, R., Vetter, J., Fang, Y. & Ong, H. (2006). IPMI-based Efficient Notification Framework for Large Scale Cluster Computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*.

Singapore: IEEE, pp. 23. doi: https://doi-org.libraryproxy.his.se/10.1109/CCGRID.2006.1630918

Modig, D. (2019). *Introduction to Ethical Hacking*. [PowerPoint presentation]. Skövde: Högskolan i Skövde.

Moore, H. (2013). A Penetration Tester's Guide to IPMI And Bmcs. *Rapid7 Blog*[blog], July 2. https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/ [2020-04-10]

Nagy, R., Christensen, T. & Horne, G. (2019). *Cybersecurity Automation for Dummies*. Hobooken: John Wiley & Sons, Inc.

Roemer, T., 2020. *Dell/Idrac-Redfish-Scripting*. https://github.com/dell/iDRAC-Redfish-Scripting/blob/master/Redfish%20Python/ImportSystemConfigurationLocalFilenameREDFISH.py [2020-05-12]

White, T., Litkey, J. & Calvert, D. (2005). Design of an Autonomic Element for Server Management. In *Second International Conference on Autonomic Computing (ICAC'05)*. Seattle: IEEE, pp. 147-158. doi: https://doi-org.libraryproxy.his.se/10.1109/ICAC.2005.23

Witte, G., Cook, M., Kerr, M. & Shaffer, S. (2012). *Security Automation Essentials*. New York: McGraw-Hill Education.

Appendix A OME Discovery Script

```
#!/usr/bin/python


import requests as req
import re
#disable https insecure warnings as we do not verify those
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# read username and pass for OME from a file called omedisc.conf
confFile = open('omedisc.conf', 'r')
line = confFile.readlines()

user = ""
passwd = ""

#read the conf file for the credentials
for row in line:
    row = row.strip()
    array = row.split('=')
    if array[0] == "User":
        user = array[1]
    elif array[0] == "Pass":
        passwd = array[1]

#get machines in the query group
response = req.get("https://1.1.1.1/api/GroupService/Groups(10535)/Devices", auth=(user, passwd),
verify=False)

#parse out the IDs in OME
ids = re.findall(r'\(\d{5}', response.text)

queryGroup = []
serverToAdd = []
for i in ids:
    #remove unnessessary paranthesis
    i = i.replace('(', '')
    if i not in queryGroup:
        queryGroup.append(i)

#get the devices in the static group to compare if something is missing
response = req.get("https://1.1.1.1/api/GroupService/Groups(10182)/Devices", auth=(user, passwd),
verify=False)
#regex match the IDs
ids = re.findall(r'\(\d{5}', response.text)
staticGroup = []
for i in ids:
    #same as before
    i = i.replace('(', '')
    #check so that we do only have unique IDs in the list
    if i not in staticGroup:
        staticGroup.append(i)

for new in queryGroup:
    #if they are not in static group but in query we mark that ID for insertion in OME
    if new not in staticGroup:
        serverToAdd.append(new)

# create the payload
payload = {'GroupId':10182, 'MemberDeviceIds':serverToAdd}
#issue the command to add to the static group
req.post(url='https://1.1.1.1/api/GroupService/Actions/GroupService.AddMemberDevices', json=payload,
auth=(user, passwd), verify=False)
```

Appendix B iDRAC Configuration Script

```
#!/bin/python

import requests
import json
import xml.etree.cElementTree as ET
import os
import subprocess
import re
import socket
import struct
#for benchmarking
from datetime import datetime

begin_time = datetime.now()

#change depending on implementation 2 is NFS share and 3 is using redfish API. 4 for the untested, proposed method
implementation = 3
#if implementation is 4 then we seperate if the script is generating or pushing
isGenerating= True

#this should be all cases but when using implementation 4 and pushing
if isGenerating == True:
    #get the API token to be used for DB calls
    token = open("token.conf", 'r').read().strip()

    #this will search for certain machines
    url = "https://someDatabase.com/api/5.0/ci/search"

    #payload is what params we are searching for
    payload = "{\"Some_Value\":\"54\",\"Some_Other\":\"74\"}"
    headers = {
            'Content-Type': 'application/json',
              'Accept': 'application/json',
                'Authorization': token
                }

    #grab the response which contains the matched machines
    response = requests.request("POST", url, headers=headers, data = payload)
    #make the results in a json format to easier get the results and only look for the result part, aka discard the paginator
    objects = (json.loads(response.text.encode('utf8'))['result'])
    EmptyCI = []
    PreppedCI = []
    for element in objects:
        #if description field is empty then we add it to the empty list to prepare the field
        if element['description'] is None:
            EmptyCI.append(element)
        elif element['description'].startswith('Configured:'):
            #check that the field is configured as the script needs and if so add them to the list for machines to configure
            if element['description'].split(':')[1] == "false":
                PreppedCI.append(element)

    #below is for changing values of an CI
    for machine in EmptyCI:
        url = "https://someDatabase.com/api/5.0/ci/" + str(machine['id'])
        #prepare the description field to be empty but with the right formatting
        payload = "{\"description\":\"Configured:false\"}"
        headers = {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
        'Authorization': token
        }

        response = requests.request("PUT", url, headers=headers, data = payload)

    ServiceTagIP = {}
```

```
    #check that prepped CI is not empty
    if len(PreppedCI) > 0 and implementation == 3:
        #time to check what IP of iDRAC has what servertag
        for i in range(181, 190):
            #ping the host so that we only curl machines that are on
            pingCommand = "ping -c1 1.1.1." + str(i) + " >/dev/null 2>&1"
            if (os.system(pingCommand)) == 0:
                #curl the machines to be able to detect what servicetag is on what ip (from DHCP)
                curl = subprocess.check_output("curl -k -G https://1.1.1." + str(i) + "/redfish/v1/", shell=True)
                if "ServiceTag" in curl:
                    ServiceTagIP[curl[curl.find('"ServiceTag":"')+14:].split('"')[0]] = "1.1.1." + str(i)

    #for making the xml config files
    for machine in PreppedCI:
        url = "https://someDatabase.com/api/5.0/ip/search"
        hostname = machine['hostname']
        payload = "{\"dns_entry\":\"" + hostname + "\"}"
        headers = {
            'Content-Type': 'application/json',
            'Accept': 'application/json',
              'Authorization': token,
                'Content-Type': 'text/plain'
                }

        response = requests.request("POST", url, headers=headers, data = payload)
        IPDatabase = (json.loads(response.text.encode('utf8'))['result'])
        #this part is dumb but as we are technically searching we get a list of responses
        FutureIP = IPDatabase[0]['ip_addr']
        NetworkID = IPDatabase[0]['network_id']
        ServiceTag = machine['serial_no']
        url = "https://someDatabase.com/api/5.0/network/search"

        payload = "{\"id\":\"" + str(NetworkID) + "\"}"
        headers = {
            'Content-Type': 'application/json',
            'Accept': 'application/json',
              'Authorization': token,
                'Content-Type': 'text/plain'
                }

        response = requests.request("POST", url, headers=headers, data = payload)
        NetDatabase = (json.loads(response.text.encode('utf8'))['result'])

        NetStart = NetDatabase[0]['ipv4_start']
        CIDR = NetDatabase[0]['ipv4_cidr']
        #need to convert CIDR to netmask for iDRACs still
        host_bits = 32 - CIDR
        FutureNetmask = socket.inet_ntoa(struct.pack('!I', (1 << 32) - (1 << host_bits)))
        NetParts = NetStart.split('.')
        LastBit = int(NetParts[3]) + 1
        FutureGateway = str(NetParts[0]) + "." + str(NetParts[1]) + "." + str(NetParts[2]) + "." + str(LastBit)

        #create the XML file
        SystemConfiguration = ET.Element('SystemConfiguration')
        Component = ET.SubElement(SystemConfiguration, 'Component', FQDD="iDRAC.Embedded.1")
        ET.SubElement(Component, "Attribute", Name="IPv4.1#Enable").text = "Enabled"
        ET.SubElement(Component, "Attribute", Name="IPv4.1#DHCPEnable").text = "Disabled"
        ET.SubElement(Component, "Attribute", Name="IPv4Static.1#Address").text = FutureIP
        ET.SubElement(Component, "Attribute", Name="IPv4Static.1#Netmask").text = FutureNetmask
        ET.SubElement(Component, "Attribute", Name="IPv4Static.1#Gateway").text = FutureGateway
        ET.SubElement(Component, "Attribute", Name="IPv4Static.1#DNS1").text = "1.1.1.1"
        ET.SubElement(Component, "Attribute", Name="IPv4Static.1#DNS2").text = "1.1.1.2"

        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#Enable").text = "Enabled"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#Schema").text = "Standard Schema"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#DomainController1").text = "1.1.1.3"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#DomainController2").text = "1.1.1.4"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#GlobalCatalog1").text = "1.1.1.3"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#GlobalCatalog2").text = "1.1.1.4"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#CertValidationEnable").text =
"Disabled"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#GCLookupEnable").text = "Disabled"
```

```
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#DCLookupEnable").text = "Disabled"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#GCRootDomain").text =
"someorg.com"
        ET.SubElement(Component, "Attribute", Name="ActiveDirectory.1#DCLookupByUserDomain").text =
"Enabled"
        ET.SubElement(Component, "Attribute", Name="ADGroup.5#Name").text = "agroup"
        ET.SubElement(Component, "Attribute", Name="ADGroup.5#Domain").text = "someorg.com"
        ET.SubElement(Component, "Attribute", Name="ADGroup.5#Privilege").text = "511"
        ET.SubElement(Component, "Attribute", Name="UserDomain.1#Name").text = "someorg.com"
        ET.SubElement(Component, "Attribute", Name="NTPConfigGroup.1#NTP1").text = "ntp.someorg.com"
        ET.SubElement(Component, "Attribute", Name="Users.3#UserName").text = "test2020"
        ET.SubElement(Component, "Attribute", Name="Users.3#Privilege").text = "511"
        ET.SubElement(Component, "Attribute", Name="Users.3#IpmiLanPrivilege").text = "Administrator"
        ET.SubElement(Component, "Attribute", Name="Users.3#Enable").text = "Enabled"
        ET.SubElement(Component, "Attribute", Name="Users.3#SolEnable").text = "Enabled"
        ET.SubElement(Component, "Attribute", Name="Users.3#ProtocolEnable").text = "Disabled"
        ET.SubElement(Component, "Attribute", Name="Users.3#AuthenticationProtocol").text = "SHA"
        ET.SubElement(Component, "Attribute", Name="Users.3#PrivacyProtocol").text = "AES"
        ET.SubElement(Component, "Attribute", Name="Users.3#SHA256Password").text =
"4699F99853C51DA1BA9559D5CAD25C74DB4C4DB4980A1C960CE9C50E24ADE738"
        ET.SubElement(Component, "Attribute", Name="Users.3#SHA1v3Key").text =
"36ebf9393d33594d99af7548ad8bce8d12a199ea"
        ET.SubElement(Component, "Attribute", Name="Users.3#MD5v3Key").text =
"4df181fb93f8525e4ff973ab590c686f"
        ET.SubElement(Component, "Attribute", Name="Users.3#SHA256PasswordSalt").text =
"C65A6070200CDD6289C9B69ED01DD771"

        tree = ET.ElementTree(SystemConfiguration)
        tree.write(ServiceTag + "-config.xml")
        okToUpdateDB = False

        #if using NFS share then move file to NFS share and delete from here
        if implementation == 2:
            okToUpdateDB = True
            subprocess.call(["scp", "-i", "/home/someplace/id_rsa", ServiceTag + "-config.xml",
"test@1.1.1.8:/srv/nfs/"])

        #if pushing directly to iDRAC instead then push file to iDRAC and delete local file
        if implementation == 3 and machine['serial_no'] in ServiceTagIP:
            okToUpdateDB = True
            exportString = "python ImportSystemConfigurationLocalFilenameREDFISH.py -ip " +
ServiceTagIP[ServiceTag] + " -u test2020 -p Syp9393 -t IDRAC -f " + ServiceTag + "-config.xml"
            os.system(exportString)

        #fourth undocumented method though more may be needed when creating the xml file as the iDRAC is
unconfigured and clean
        if implementation == 4:
            okToUpdateDB = True

        if implementation != 4:
            os.remove(ServiceTag + "-config.xml")

        os.remove(ServiceTag + "-config.xml")
        #change hydra so that configured gets set to true and rest is same
        url = "https://someDatabase.com/api/5.0/ci/" + str(machine['id'])

        payload = "{\"description\":\"Configured:true\"}"
        headers = {
          'Content-Type': 'application/json',
          'Accept': 'application/json',
          'Authorization': token
        }

        if okToUpdateDB == True:
            response = requests.request("PUT", url, headers=headers, data = payload)

        time_taken = datetime.now() - begin_time
        print("script took {}".format(time_taken))

#this should only happen if implementation is 4 and we want to push
elif implementation == 4 and isGenerating == False:
    curl = subprocess.check_output("curl -k -G https://192.168.0.120/redfish/v1/", shell=True)
```

```
    if "ServiceTag" in curl:
        ServiceTag = curl[curl.find('"ServiceTag":"')+14:].split('"')[0]
        exportString = "python ImportSystemConfigurationLocalFilenameREDFISH.py -ip 192.168.0.120 -u admin
-p calvin -t IDRAC -f " + ServiceTag + "-config.xml"
        os.system(exportString)
        os.remove(ServiceTag + "-config.xml")
```

Appendix C DHCP Options

```
option myname code 43 = text;
subnet 192.168.0.0 netmask 255.255.0.0
{
   option routers          192.168.0.1;
   option subnet-mask      255.255.255.0;
   option nis-domain       "domain.org";
   option domain-name      "domain.org";
   option domain-name-servers  192.168.1.1;
   option time-offset      -18000; # Eastern Standard Time


   option vendor-class-identifier "iDRAC";
   set vendor-string = option vendor-class-identifier;
   option myname -i 192.168.0.130 -u user -p password -n cifs -s 2 -d 0 -t 500";


   range dynamic-bootp 192.168.0.128 192.168.0.254;
   default-lease-time 21600;
   max-lease-time 43200;
}
```

Appendix D iDRAC REDFISH Import XML Script

```
#
# ImportSystemConfigurationLocalFilenameREDFISH. Python script using Redfish API to import system
configuration profile attributes locally from a configuration file.
#
# _author_ = Texas Roemer <Texas_Roemer@Dell.com>
# _version_ = 11.0
#
# Copyright (c) 2017, Dell, Inc.
#
# This software is licensed to you under the GNU General Public License,
# version 2 (GPLv2). There is NO WARRANTY for this software, express or
# implied, including the implied warranties of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE. You should have received a copy of GPLv2
# along with this software; if not, see
# http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt.
#

import requests, json, sys, re, time, warnings, argparse

from datetime import datetime

warnings.filterwarnings("ignore")

parser=argparse.ArgumentParser(description="Python script using Redfish API to import the host server
configuration profile locally from a configuration file.")
parser.add_argument('-ip',help='iDRAC IP address', required=True)
parser.add_argument('-u', help='iDRAC username', required=True)
parser.add_argument('-p', help='iDRAC password', required=True)
parser.add_argument('script_examples',action="store_true",help='ImportSystemConfigurationLocalFilenameRE
DFISH.py -ip 192.168.0.120 -u root -p calvin -t ALL --filename SCP_export_R740, this example is going to
import SCP file and apply all attribute changes for all components.
\nImportSystemConfigurationLocalFilenameREDFISH.py -ip 192.168.0.120 -u root -p calvin -t BIOS --filename
R740_scp_file -s Forced, this example is going to only apply BIOS changes from the SCP file along with forcing
a server power reboot.')
parser.add_argument('-t', help='Pass in Target value to set component attributes. You can pass in \"ALL" to set
all component attributes or pass in a specific component to set only those attributes. Supported values are:
ALL, System, BIOS, IDRAC, NIC, FC, LifecycleController, RAID.', required=True)
parser.add_argument('-s', help='Pass in ShutdownType value. Supported values are Graceful, Forced and
NoReboot. If you don\'t use this optional parameter, default value is Graceful. NOTE: If you pass in NoReboot
value, configuration changes will not be applied until the next server manual reboot.', required=False)
parser.add_argument('-f', help='Pass in Server Configuration Profile filename', required=True)
parser.add_argument('-e', help='Pass in end HostPowerState value. Supported values are On and Off. If you
don\'t use this optional parameter, default value is On', required=False)
args=vars(parser.parse_args())

idrac_ip=args["ip"]
idrac_username=args["u"]
idrac_password=args["p"]
filename=args["f"]

try:
    f=open(filename,"r")
except:
    print("\n-FAIL, \"%s\" file doesn't exist" % filename)
    sys.exit()

url =
'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemConfiguration' % idrac_ip

# Code needed to modify the XML to one string to pass in for POST command
z=f.read()
z=re.sub(" \n ","",z)
z=re.sub(" \n","",z)
xml_string=re.sub("    ","",z)
f.close()

payload = {"ImportBuffer":"","ShareParameters":{"Target":args["t"]}}
```

```
if args["s"]:
    payload["ShutdownType"] = args["s"]
if args["e"]:
    payload["HostPowerState"] = args["e"]

payload["ImportBuffer"]=xml_string
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False,
auth=(idrac_username, idrac_password))

#print '\n- Response status code is: %s' % response.status_code


d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print("\n- FAIL: status code %s returned" % response.status_code)
    print("- Detailed error information: %s" % d)
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print("\n- FAIL, status code not 202\n, code is: %s" % response.status_code )
    sys.exit()
else:
    print("\n- %s successfully created for ImportSystemConfiguration method\n" % (job_id) )

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()


start_time=datetime.now()
while True:
    #req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id),
auth=(idrac_username, idrac_password), verify=False)
    count = 1
    while True:
        if count == 5:
            print("- FAIL, 5 attempts at getting job status failed, script will exit")
            sys.exit()
        try:
            req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id),
auth=(idrac_username, idrac_password), verify=False)
            break
        except RuntimeError as error_message:
            print("- FAIL, requests command failed to GET job status, detailed error information: \n%s" %
error_message)
            error_message = str(error_message)
            if "Failed to establish a new connection" in error_message:
                print("- WARNING, failed to establish connection, executing command again")
                time.sleep(10)
                count+=1
                continue
            else:
                sys.exit()
    statusCode = req.status_code
    data = req.json()
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        pass
        time.sleep(3)
    else:
        print("Query job ID command failed, error code is: %s" % statusCode)
        sys.exit()
    if "failed" in data['Oem']['Dell']['Message'] or "completed with errors" in data['Oem']['Dell']['Message'] or
"Not one" in data['Oem']['Dell']['Message'] or "not compliant" in data['Oem']['Dell']['Message'] or "Unable" in
data['Oem']['Dell']['Message'] or "The system could not be shut down" in data['Oem']['Dell']['Message'] or "No
device configuration" in data['Oem']['Dell']['Message'] or "timed out" in data['Oem']['Dell']['Message']:
```

```
        print("- FAIL, Job ID %s marked as %s but detected issue(s). See detailed job results below for more
information on failure\n" % (job_id, data['Oem']['Dell']['JobState']))
        print("- Detailed configuration changes and job results for \"%s\"\n" % job_id)
        try:
            for i in data["Messages"]:
                for ii in i.items():
                    if ii[0] == "Oem":
                        for iii in ii[1]["Dell"].items():
                            print("%s: %s" % (iii[0], iii[1]))
                    else:
                        print("%s: %s" % (ii[0], ii[1]))
                print("\n")
        except:
            print("- FAIL, unable to get configuration results for job ID, returning only final job results\n")
            for i in data['Oem']['Dell'].items():
                print("%s: %s" % (i[0], i[1]))

            print("- %s completed in: %s" % (job_id, str(current_time)[0:7]))
        sys.exit()

    elif "No reboot Server" in data['Oem']['Dell']['Message']:
        print("- PASS, job ID %s successfully marked completed. NoReboot value detected and config changes
will not be applied until next manual server reboot\n" % job_id)
        print("\n- Detailed job results for job ID %s\n" % job_id)
        for i in data['Oem']['Dell'].items():
            print("%s: %s" % (i[0], i[1]))
        sys.exit()
    elif "Successfully imported" in data['Oem']['Dell']['Message'] or "completed with errors" in
data['Oem']['Dell']['Message'] or "Successfully imported" in data['Oem']['Dell']['Message']:
        print("- PASS, job ID %s successfully marked completed\n" % job_id)
        print("- Detailed configuration changes and job results for \"%s\"\n" % job_id)
        try:
            for i in data["Messages"]:
                for ii in i.items():
                    if ii[0] == "Oem":
                        for iii in ii[1]["Dell"].items():
                            print("%s: %s" % (iii[0], iii[1]))
                    else:
                        print("%s: %s" % (ii[0], ii[1]))
                print("\n")
        except:
            print("- FAIL, unable to get configuration results for job ID, returning only final job results\n")
            for i in data['Oem']['Dell'].items():
                print("%s: %s" % (i[0], i[1]))

        print("- %s completed in: %s" % (job_id, str(current_time)[0:7]))
        sys.exit()

    elif "No changes" in data['Oem']['Dell']['Message'] or "No configuration changes" in
data['Oem']['Dell']['Message']:
        print("\n- PASS, job ID %s marked completed\n" % job_id)
        print("- Detailed job results for job ID %s\n" % job_id)
        for i in data['Oem']['Dell'].items():
            print("%s: %s" % (i[0], i[1]))
        sys.exit()
    else:
        print("- WARNING, JobStatus not completed, current status: \"%s\", percent complete: \"%s\"" %
(data['Oem']['Dell']['Message'],data['Oem']['Dell']['PercentComplete']))
        time.sleep(1)
        continue
```

Appendix E Average time calculation script

```
#!/bin/python
from datetime import timedelta

#OME Discovery data
data11 = ["0:03:17",
"0:03:25",
"0:03:24",
"0:03:34",
"0:03:22",
"0:03:25",
"0:03:23",
"0:03:33",
"0:03:32",
"0:03:30"]

#OME Push data
data12 = ["0:00:41",
"0:00:37",
"0:00:55",
#failed here
"0:00:57",
"0:00:46",
"0:01:31",
"0:01:19",
"0:00:39",
"0:00:39"]

#OME total data
data15 = ["0:03:58",
"0:04:02",
"0:04:19",
#disregarding this as it was a fail on push
"0:04:19",
"0:04:11",
"0:04:54",
"0:04:52",
"0:04:11",
"0:04:09"]

#NFS generation data
data2 = ["0:00:09.084279",
"0:00:05.662361",
"0:00:03.821112",
"0:00:04.694050",
"0:00:03.795095",
"0:00:04.801468",
"0:00:04.379154",
"0:00:05.040183",
"0:00:06.246946",
"0:00:05.695747"]

#Direct Push data
data3 = ["0:01:38.451352",
"0:01:36.364070",
"0:01:42.028054",
"0:01:36.950972",
"0:01:36.194820",
"0:01:35.648945",
"0:01:35.642729",
"0:01:41.211671",
"0:01:37.098824",
"0:01:38.609861"]

def converter(x):
    if '.' not in x:
        x += '.000000'
    hrs, mins, secs, millis = map(int, x[:-3].replace('.', ':').split(':'))
    return timedelta(hours=hrs, minutes=mins, seconds=secs, milliseconds=millis)
```

```
res = sum(map(converter, data11), timedelta(0)) / len(data11)
print("OME Discovery took average of ", res)
res = sum(map(converter, data12), timedelta(0)) / len(data12)
print("OME Push took average of ", res)
res = sum(map(converter, data15), timedelta(0)) / len(data15)
print("OME TOTAL took average of ", res)
res = sum(map(converter, data2), timedelta(0)) / len(data2)
print("NFS took average of ", res)
res = sum(map(converter, data3), timedelta(0)) / len(data3)
print("Direct took average of ", res)
```