508.1

Advanced Incident Response and Threat Hunting

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, USER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree, User may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There’s an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

All reference links are operational in the browser-based delivery of the electronic workbook.

This page intentionally left blank.

This page intentionally left blank.

Exercise 0

Before Class Begins: VM Installation

This page intentionally left blank.

Exercise 1.1

APT Intrusion Scenario
Before Class Begins: Read Scenario

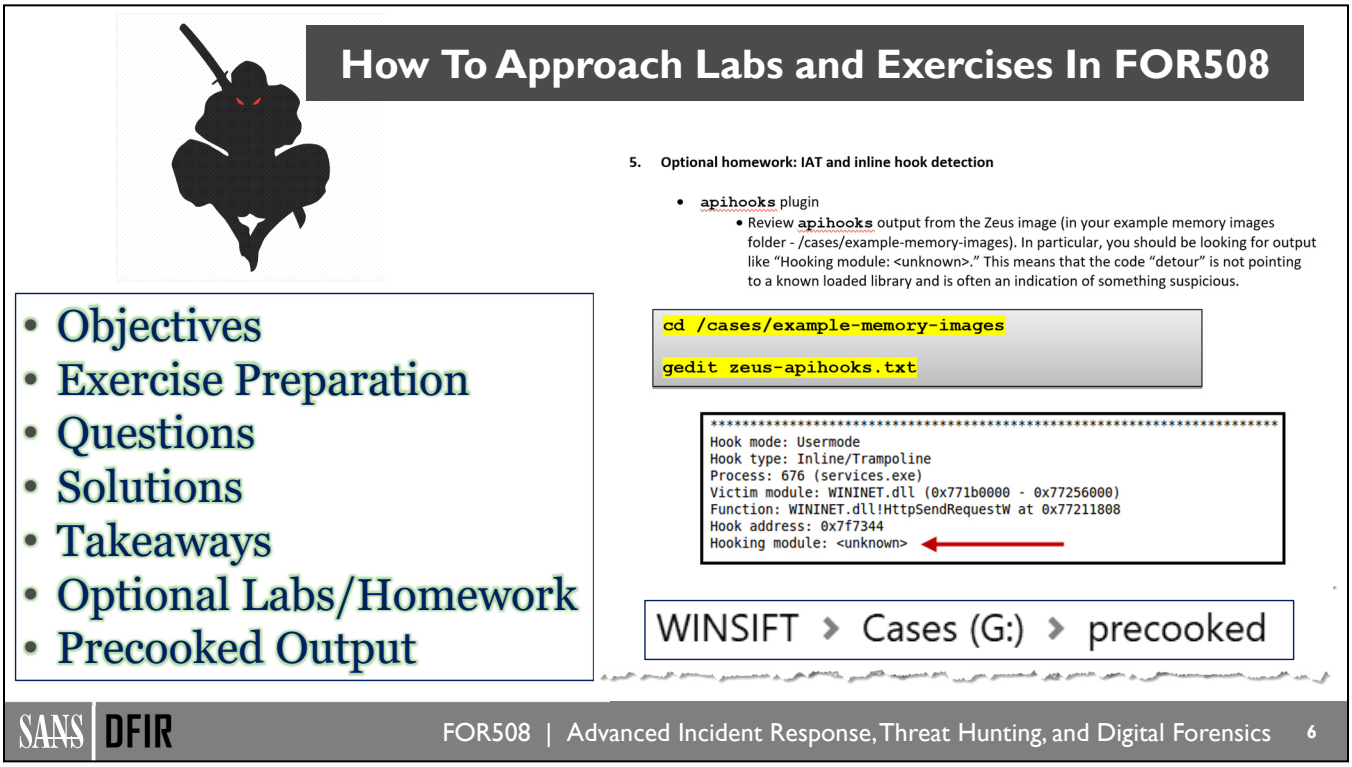
This page intentionally left blank.

Exercise 1.2

Mounting Triage VHDX Evidence

Average Time: 10 Minutes

This page intentionally left blank.



The FOR508 SRL Intrusion Exercise Workbook is full of crucial information that will assist with course objectives and provide guidelines and instructions for many investigations in the future.

To ensure you get the most out of each lab, we would like to step you through the different sections of the workbook. The workbook is specifically designed to enable students from a variety of backgrounds and different skill levels to get the most out of each lab.

Exercise Objectives:
This section provides the big picture of what the exercise is meant to show or teach. We might be demonstrating an analytical technique or the specific output of a forensic tool. We strongly recommend you quickly look over these objectives when beginning the exercise.

Exercise Preparation:
Exercises are designed to stand on their own. This allows students who are reviewing the exercises to jump in without necessarily having completed previous exercises. We typically outline the specific system, the condition of that system, or the capabilities that must be enabled before moving into the actual exercise. Skipping over this step could mean that your system might not be ready for analysis.

Questions without Explanations and Questions with Step-by-Step Instructions:
We want you to focus on the core concepts and analytical techniques of each exercise instead of just running blindly through a tool. Eventually, you will master the tool, but the most important part of this course, especially if you are new, is to focus on the output of the tool and how to properly analyze it.
There are two parts to every exercise:

1. Exercise questions without any help or explanations.
2. Solutions with full step-by-step instructions and explanations.

For most students doing the exercise for the first time, we recommend liberally using the solutions to see the step-by-step instructions and explanations.

Think of the exercises as being accomplished in three ways. FOR508 is an advanced course, so we recommend beginning or intermediate students approach exercises as follows:

Gain familiarity: The first time through the exercise students should liberally use the solutions to see the step-by-step instructions and familiarize themselves with the overall topic and techniques. Remember you are here to learn, not to fight your system or become confused. You will get more from the exercise by following along and mimicking what you see directly while reading the full (and sometimes lengthy) explanations.

Gain mastery: When reviewing the exercise again, we recommend using the "hybrid" approach. This approach has you start with the part of the exercise that has questions without any help or explanations, but then reference the solutions when you get stuck. Students should be comfortable doing the exercises themselves by the time they reach the final capstone exercise, and the capstone systems will rely on the same procedures and techniques found in the exercises to provide more experience.

Achieve mastery: Once you can complete the exercise without referencing the step-by-step information within the solutions, you have mastered the skill. This is a great way to demonstrate you are ready to pass the certification for this course. If you have already mastered the skills on exercises from the start, it is likely you have learned those skills already or know them from previous courses. It is also likely you already have the skills needed to do Incident Response and Threat Hunting in the real world. Many students take a class to obtain new skills, but the more advanced you get, the fewer new skills you will learn each time taking a course. We aim for students to reach this stage after having completed the final capstone exercise, reviewing the exercises a few more times, and then testing themselves by completing the full exercise without referencing the step-by-step information within the solutions.

Takeaways:

For every exercise, the takeaway section highlights important case-related artifacts we uncovered as a result of our analysis. The takeaway section is important because these artifacts will build on one another as we progress through the course. Sometimes it is hard to remember "How did we find pa.exe?" in a new exercise that suddenly asks you to use prior knowledge to look for something new. We advise regularly reviewing the takeaways from each exercise to refresh your memory of the ongoing incident we will be investigating.

Precooked Exercise Output:

For every exercise, there is a certain amount of "keyboard kung fu" necessary to complete the course. If you are struggling with the seemingly never-ending command line input, we have relevant exercise output pre-generated for you within precooked folders:
FOR508 Windows VM: **G:\precooked\<subfolder>**
Linux SIFT VM: **/cases/precooked/<subfolder>**

Using the right techniques to approach the exercises and labs from the start is essential for your success in this course. Everyone in the course is learning, so there is no reason to feel judged if you are regularly using the solutions during each exercise. Take advantage of the structure of the exercises to facilitate the maximum learning possible for your particular skill level and background. Enjoy!

FOR508: What You Will Receive

SIFT Workstation Ubuntu Version
- Installed in Exercise 0

FOR508 Windows VM
- Installed in Exercise 0

Large collection of forensic disk, memory, and triage images
- Supporting in-class exercises and additional homework

F-RESPONSE Enterprise (one dongle)
- Access physical disks and memory of remote systems via the network
- Perfect for incident response investigations and evidence collection

Course MP3 – Download via SANS Account Portal
- Available ~1 week after class

We have worked hard in this class to provide students with the most up-to-date and relevant toolkit available. You may be familiar with the SIFT Workstation, a Linux-based forensic distribution that has become a mainstay in the forensic community. The SIFT project was the brainchild of Rob Lee and is managed by Erik Kristensen on behalf of SANS and the entire forensics community. It takes hundreds of hours of work to release each version and get the hundreds of embedded tools to play nicely with one another. While you will be using the virtual machine version of the SIFT, you can now install SIFT on bare metal by installing Ubuntu Linux and then adding the "SIFT package".[1] Linux works efficiently even on older hardware, so if you have an old system collecting dust in the closet, pull it out and make it a forensic workstation!

You will also be using a brand-new version of the SIFT, with a Microsoft Windows wrapper. In this Windows virtual machine, we have embedded the Linux SIFT using the new Windows Subsystem for Linux (WSL) capability. This is an exciting leap forward, allowing both Windows and Linux tools to natively co-exist. You will have many opportunities to see the benefits in action.

The data set you are provided with in this class was purposely designed to be larger than necessary. You will have the opportunity to examine over twenty systems for host- and memory-based forensic artifacts. This leaves many systems for you to continue your learning on long after class is completed.

[1] SANS SIFT Documentation: https://for508.com/tz6ib

FOR508.1 | Advanced Incident Response, Threat Hunting, and Digital Forensics

Advanced Incident Response and Threat Hunting

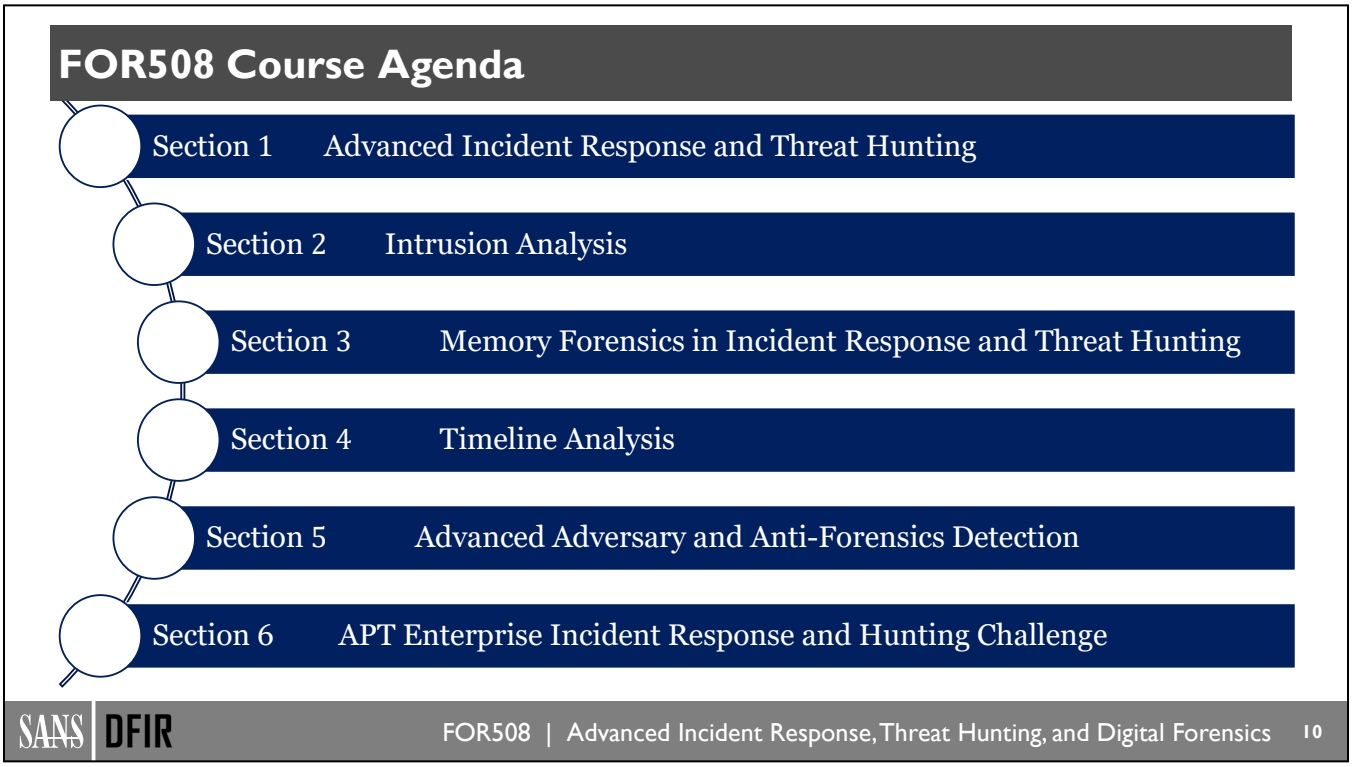
Welcome to FOR508! We are very excited to share this course with you. This is Section 1.

Author team:

Rob Lee
rlee@sans.org
https://twitter.com/robtlee
https://twitter.com/sansforensics

Chad Tilbury
ctilbury@sans.org
https://twitter.com/chadtilbury

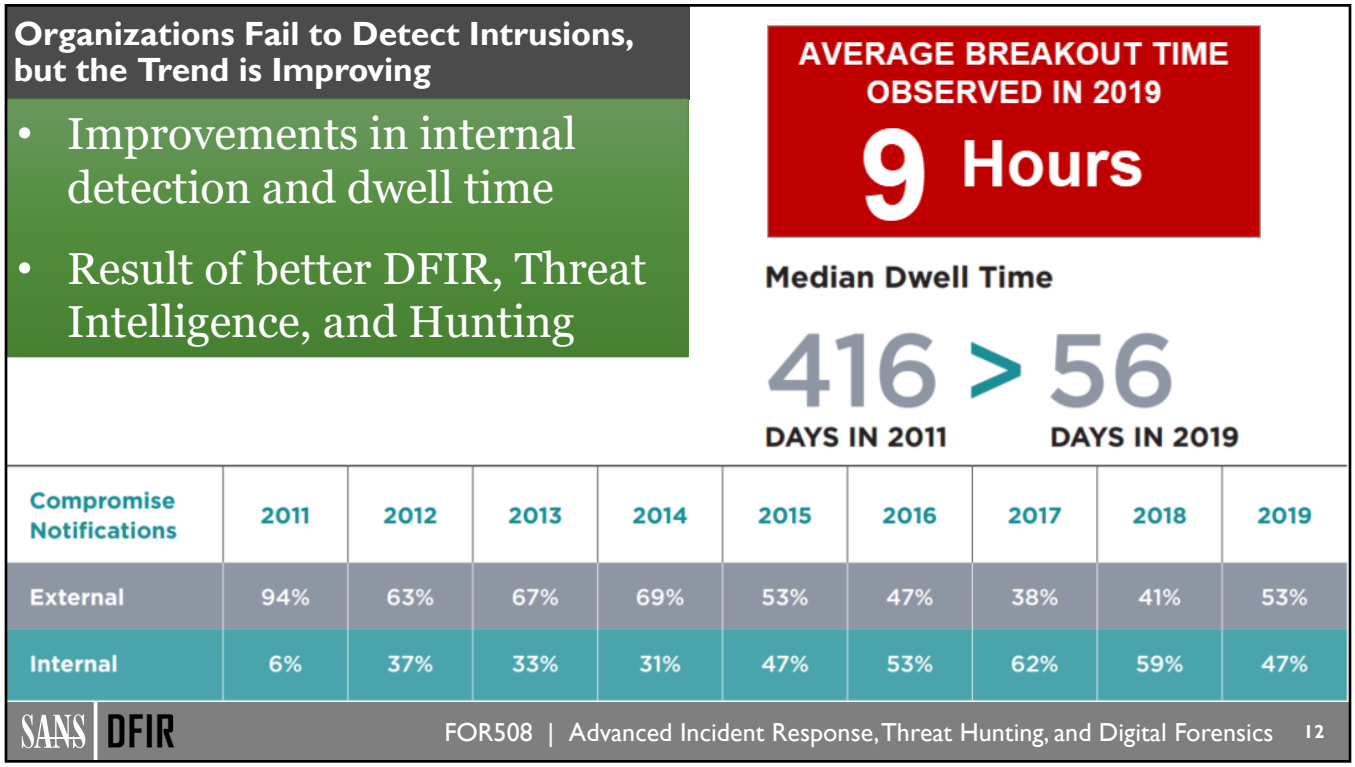
Mike Pilkington
mpilkington@sans.org
https://twitter.com/mikepilkington



This page intentionally left blank.

The Challenge of Information Security

This page intentionally left blank.



FACT: Over the last decade most organizations failed at detecting intrusions. However, we have seen dramatic improvement as organizations begin to take incident response seriously and take a more active approach in threat hunting.

The last decade has not been kind to network defenders. Our existing security models became defunct, and attackers have used the enormous complexity of enterprise networks against us. But the tide is shifting. Statistics from Mandiant M-Trends show a dramatic increase in internal detections, an excellent indicator for the increasing sophistication of network defenders.[1] Dwell time, the time an attacker has remained undetected within a network, has also been dramatically reduced.[1] Dwell time is a very important metric to track as it directly correlates with the ability of an attacker to accomplish their objectives. When an organization gets attacker dwell time down to days or weeks, they are mitigating attacks before attackers can complete their missions, with the added benefit of much less costly remediation and recovery costs. On the flip side, attackers are also becoming more sophisticated. CrowdStrike reported the 2019 average "breakout" time for attackers was nine hours (and depending on the threat actor, less than an hour).[2] Breakout time is the time it takes an intruder to begin moving laterally once they have an initial foothold in the network. As more organizations see highly targeted attacks, breakout time has been greatly reduced. It takes a mature security team with exceptional network visibility to keep pace with top level threat actors.

[1] Mandiant M-Trends: https://for508.com/br2qe
[2] CrowdStrike Global Threat Report: https://for508.com/17ljg

Threat Landscape

The Threats

- **APT**
  - **Nation-State Actors**
- **Organized Crime**
- **Hacktivists**

The Reality

- Most organizations still have a difficult time responding to intrusions from advanced adversaries

Threats to the modern enterprise are legion and defending against the hordes of attackers can seem impossible. Over the past decade, we have seen a dramatic increase in sophisticated attacks against organizations. Nation-state attacks originating from China and Russia, often referred to as Advanced Persistent Threat (APT) actors, have proved difficult to suppress. Massive financial attacks from the four corners of the globe have resulted in billions of dollars in losses. Ransomware and extortion became an existential threat almost overnight. While the odds are stacked against us, the best teams out there are proving that these threats can be managed and mitigated. The adversary is good and getting better. Are we learning how to counter them? Yes, we are.

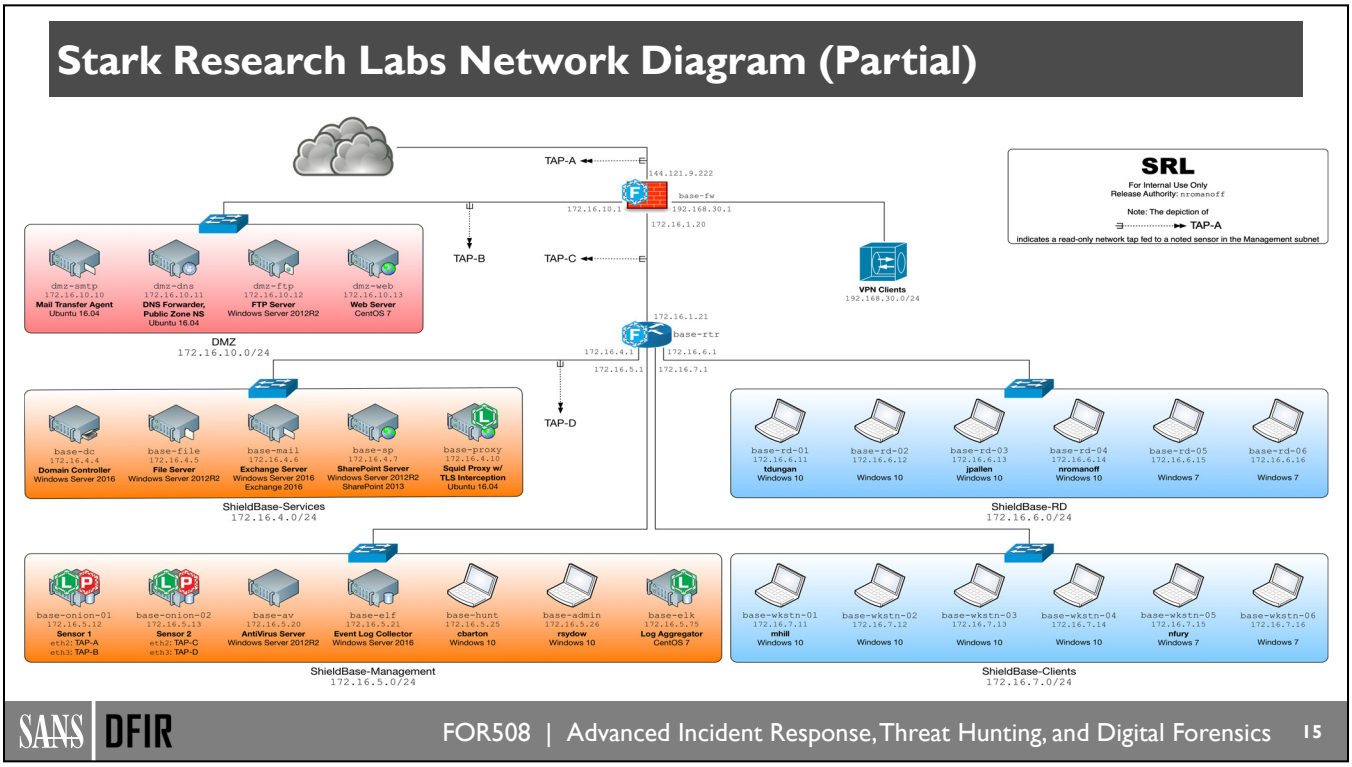
This course was designed to help organizations increase their capability to detect and respond to intrusions. This is an achievable goal and begins by teaching you the tools and techniques necessary to find evil in your network. Attackers have taken an early lead in this war, but there are many battles yet to be fought. We must get to a point where most organizations can detect their own intrusions. Unlike other threats in society, there is no one out there to come to your rescue. Law enforcement and government agencies are overwhelmed and in this crucial part of the world economy, organizations largely need to fend for themselves. This course is designed to make you and your organization an integral part of the solution. It is our hope that you can take the concepts, tools, and techniques you learn here and apply them to make a difference. Get ready to hunt!

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

SRL Intrusion Scenario
Exercise 1.1

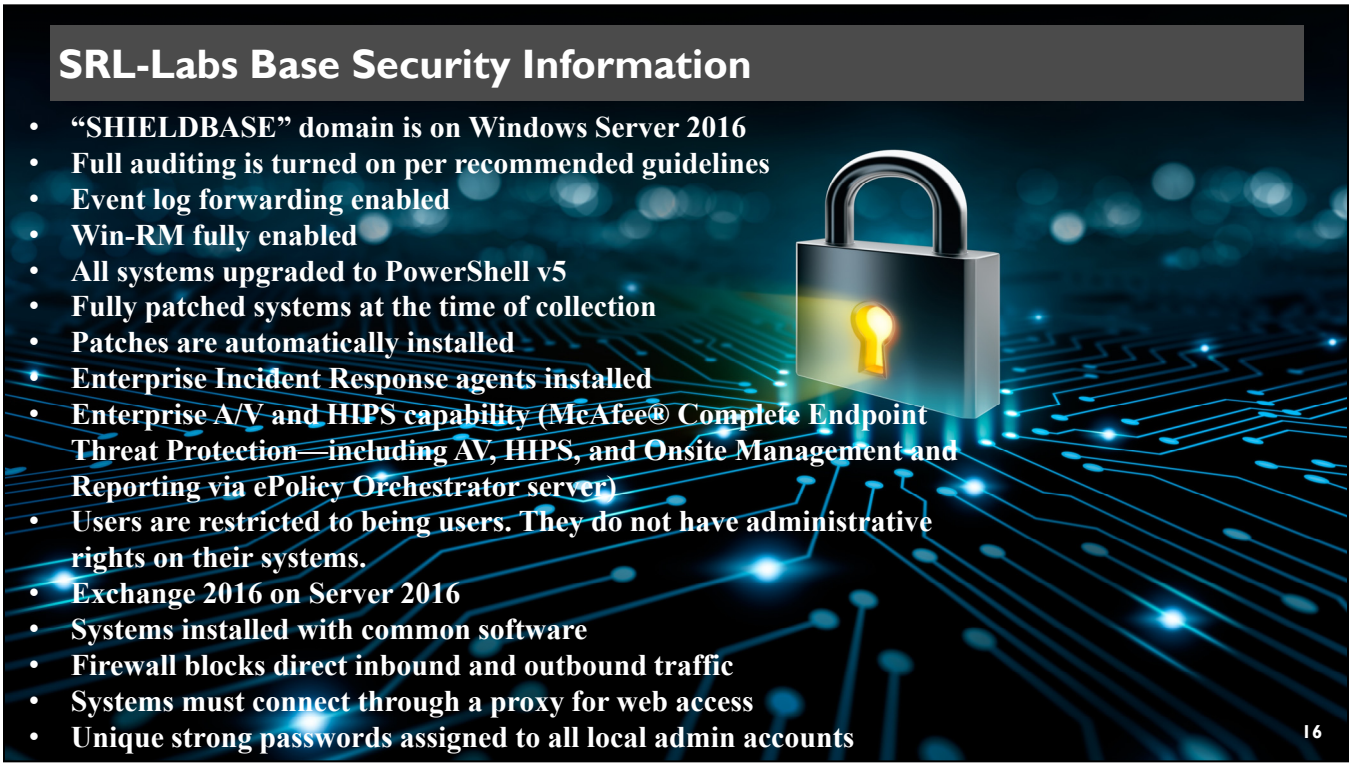
COUNTRIES TARGETED

This page intentionally left blank.



Abridged SRL Network Map. You have a copy of this in your course materials.

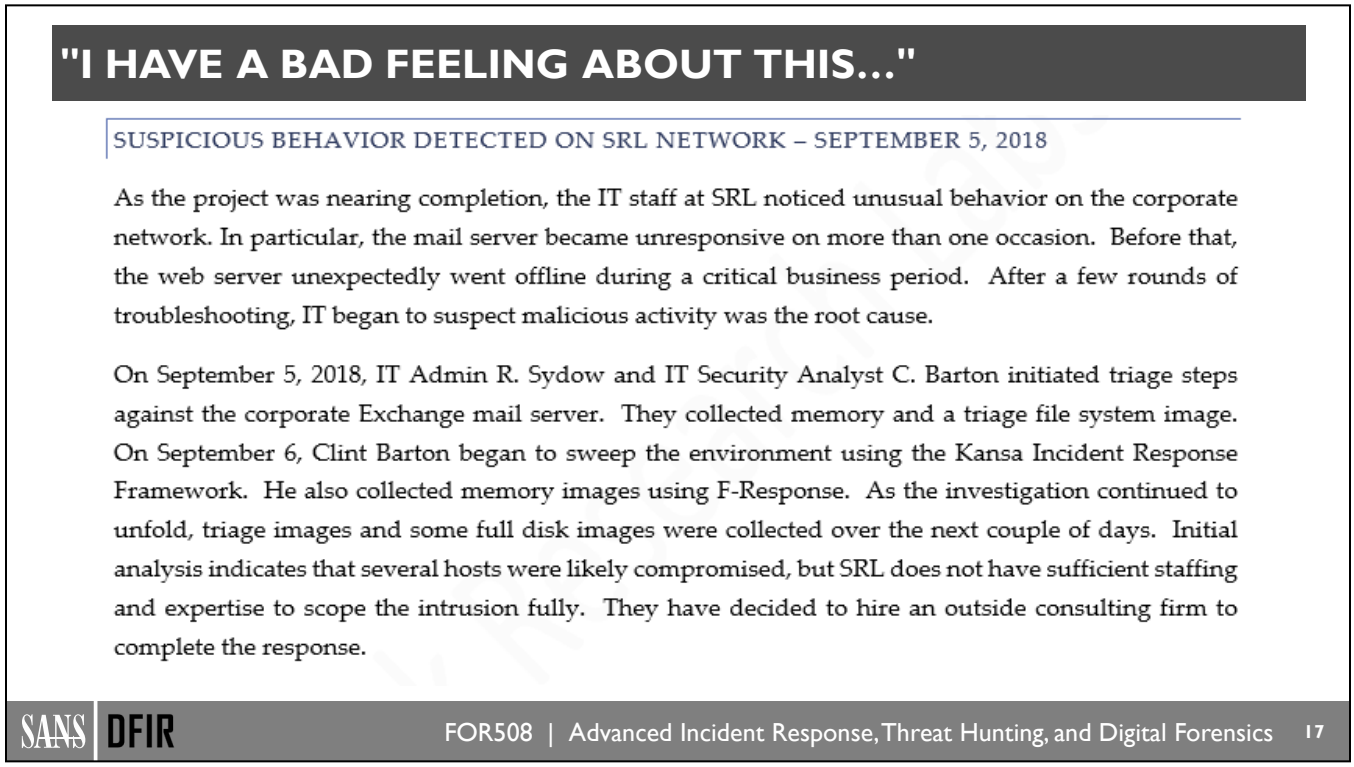
Username	Full Name	Role in SRL	Host Name	Host IP	Host OS
mhill	M Hill	CEO	base-wkstn-01	172.16.7.11	Win10
nfury	N Fury	Sr. Manager	base-wkstn-05	172.16.7.15	Win7
tdungan	T Dungan	Sr. Researcher	base-rd-01	172.16.6.11	Win10
nromanoff	N Romanoff	Program Manager	base-rd-04	172.16.6.14	Win10
rsydow	R Sydow	IT Admin User	base-admin	172.16.5.26	Win10
rsydow-a	R Sydow – Admin	Domain Admin Acct			
Cbarton	C Barton	IT Security User	base-hunt	172.16.5.25	Win10
cbarton-a	C Barton – Admin	Security Admin Acct			



The SRL network is representative of many medium-sized enterprises. It has several major internal segments separated by a single firewall with little network protection between hosts once inside of that perimeter. The network was setup according to US government guidelines, with many security controls implemented:

SRL LABS BASE DOMAIN INFORMATION

- **"SHIELDBASE" domain is on Windows Server 2016**
- **Full auditing turned on per recommended guidelines**
- **Event log forwarding enabled**
- **Win-RM fully enabled**
- **All systems upgraded to PowerShell v5**
- **Fully patched systems**
- **Patches are automatically installed**
- **Enterprise Incident Response agents installed**
- **Enterprise A/V and HIPS capability (McAfee® Complete Endpoint Threat Protection—including AV, HIPS, and Onsite Management and Reporting via ePolicy Orchestrator server)**
- **Users restricted to being users. Users do not have administrative rights on their systems.**
- **Exchange 2016 on Server 2016**
- **Systems installed with common software**
- **Firewall blocks direct inbound and outbound traffic**
- **Systems must connect through a proxy for web access**
- **Unique strong passwords assigned to all local admin accounts**



SUSPICIOUS BEHAVIOR DETECTED ON SRL NETWORK – SEPTEMBER 5, 2018
Stark Research Labs (SRL) is a government-sponsored laboratory that researches specialized metal alloys and bioengineering capabilities. Lately, SRL has been tasked to find the secret formula for Carbonadium-alloy. The lead researcher, T. Dungan, has been making progress, and it looks like, with the help of others, he was finally able to replicate the formula. T. Dungan has been working tirelessly on trying to find the missing formula for the past two years.

As the project was nearing completion, the IT staff at SRL noticed unusual behavior on the corporate network. In particular, the mail server became unresponsive on more than one occasion. Before that, the web server unexpectedly went offline during a critical business period. After a few rounds of troubleshooting, IT began to suspect malicious activity was the root cause.

On September 5, 2018, IT Admin R. Sydow and IT Security Analyst C. Barton initiated triage steps against the corporate Exchange mail server. They collected memory and a triage file system image. On September 6, Clint Barton began to sweep the environment using the Kansa Incident Response Framework. He also collected memory images using F-Response. As the investigation continued to unfold, triage images and some full disk images were collected over the next couple of days. Initial analysis indicates that several hosts were likely compromised, but SRL does not have sufficient staffing and expertise to scope the intrusion fully. They have decided to hire an outside consulting firm to complete the response.

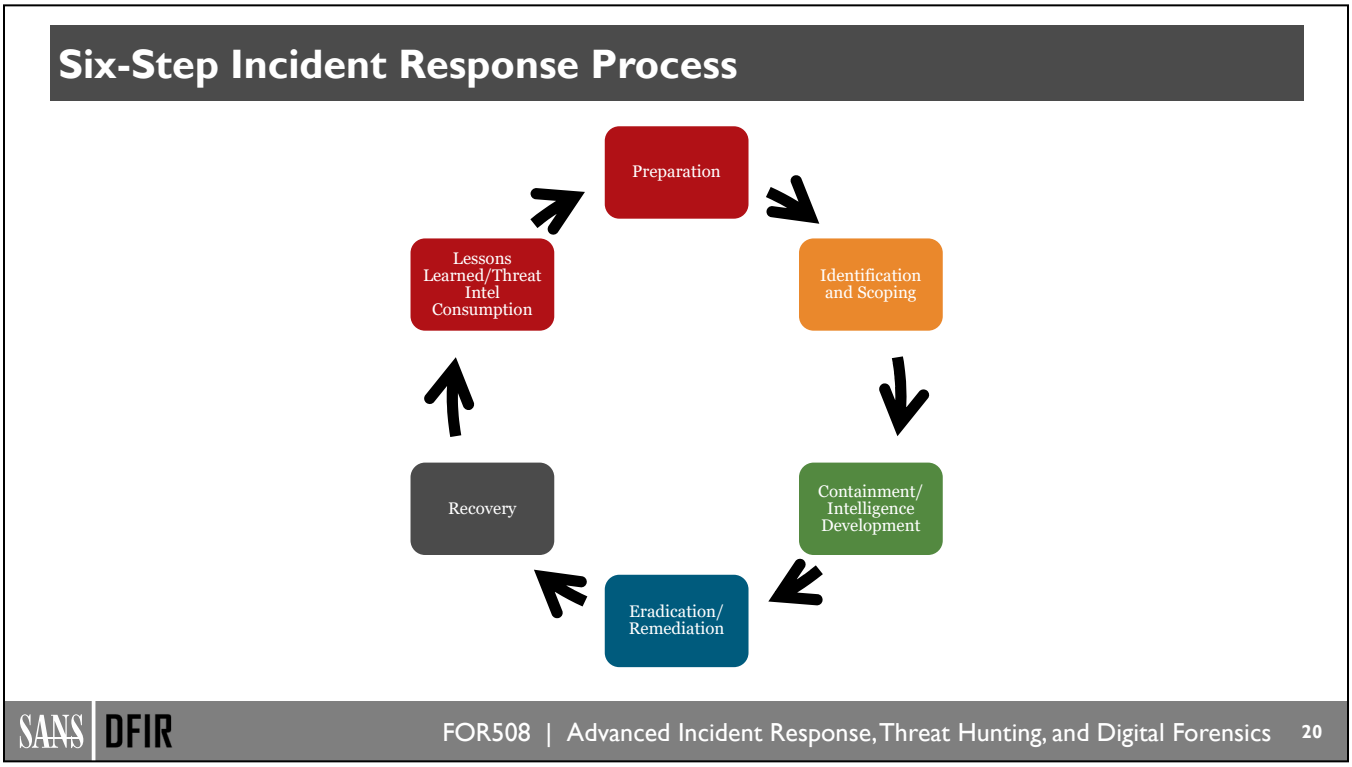
READ EXERCISE 1: SRL INTRUSION SCENARIO FOR MORE DETAILS.

This page intentionally left blank.

FOR508.1 | Advanced Incident Response, Threat Hunting, and Digital Forensics

Advanced Incident Response and Threat Hunting

This page intentionally left blank.



Computer intrusion incident response can be broken down into a six-step incident process. This model originated from United States government agencies and has been extensively documented by the US National institute for Standards and Technology (NIST).[1] It has a long and proven history of success and is a good starting place to evaluate the steps required to respond and recover from an incident.

Overview of the Six-Step Incident Response Process
The six steps of this model are preparation, identification, containment, eradication, recovery, and lessons learned.

Preparation
Incident response methodologies emphasize preparation—not only establishing a response capability so the organization is ready to respond to incidents but also preventing incidents by ensuring that systems, networks, and applications are sufficiently secure.

Identification
Identification is triggered by a suspicious event. This could be an alert from a security appliance, a call to the help-desk, or the result of something discovered via threat hunting. Event validation should occur and a decision made as to the severity of the finding (not valid events lead to a full incident response). Once an incident response has begun, this phase is used to better understand the findings and begin scoping the network for additional compromise.

Containment and Intelligence Development
In this phase, the goal is to rapidly understand the adversary and begin crafting a containment strategy. Responders must identify the initial vulnerability or exploit, how the attackers are maintaining persistence and laterally moving in the network, and how command and control is being accomplished. In conjunction with the previous scoping phase, responders will work to have a complete picture of the attack and often implement changes to the environment to increase host and network visibility. Threat intelligence is one of the key products of the IR team during this phase.

Eradication and Remediation
Arguably the most important phase of the process, eradication aims to remove the threat and restore business operations to a normal state. However, successful eradication cannot occur until the full scope of the intrusion is understood. A rush to this phase usually results in failure. Remediation plans are developed, and recommendations are implemented in a planned and controlled manner. Example changes to the environment include:

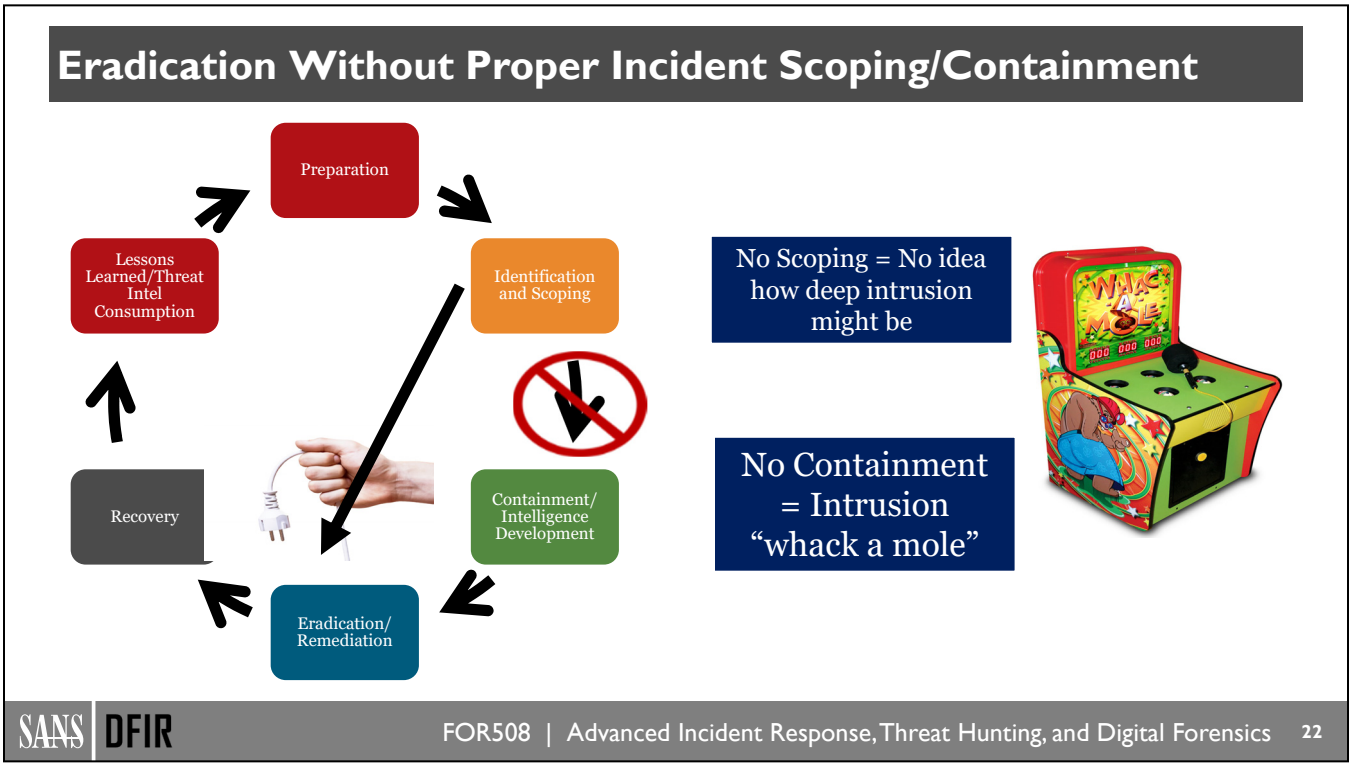
- Block malicious IP addresses
- Blackhole malicious domain names
- Rebuild compromised systems
- Coordinate with cloud and service providers
- Enterprise-wide password changes
- Implementation validation

Recovery
Recovery leads the enterprise back to day-to-day business. The organization will have learned a lot during the incident investigation and will invariably have many changes to implement to make the enterprise more defensible. Recovery plans are typically divided into near-, mid-, and long-term goals, and near-term changes should start immediately. The goal during this phase is to improve the overall security of the network and to detect and prevent immediate reinfection. Some recovery changes could include:

- Improve Enterprise Authentication Model
- Enhanced Network Visibility
- Establish Comprehensive Patch Management Program
- Enforce Change Management Program
- Centralized Logging (SIM/SIEM)
- Enhance Password Portal
- Establish Security Awareness Training Program
- Network Redesign

Follow-Up
Follow-up is used to verify the incident has been mitigated, the adversary has been removed, and additional countermeasures have been implemented correctly. This step combines additional monitoring, network sweeps looking for new breaches, and auditing the network (penetration tests and compliance) to ensure new security mechanisms are in place and functioning normally.

[1] NIST Computer Security Incident Handling Guide: http://for508.com/d0yft



The Problem? Immediate Eradication Without Proper Incident Scoping/Containment
A significant problem with the six-step incident response process is few teams follow the process as prescribed. There is often intense pressure leading to a tendency to skip immediately to the "Eradication/Remediation" phase before true scoping and understanding of the incident has occurred. While this makes it possible to begin remediation quickly, what exactly is being fixed? Moving to eradication too early removes the benefits and capabilities provided by cyber threat intelligence and intelligence-driven incident response doctrine.

A search of the NIST *Computer Security Incident Handling Guide* for the words "power," "plug," "reinstall," or "pulling" resulted in zero hits being found. Many organization's incident response policies trend toward immediate eradication through pulling the plug as their stop gap maneuver to prevent the additional spread of an attacker inside their environment. Similar methods include blocking IP addresses, rebuilding systems, and disabling compromised user accounts. While these actions prevent further attacks from that vector, they are unlikely to lead to full eradication of the attacker. The problem lies in the fact that most organizations cannot detect an intrusion early enough to detect the initial foothold and prevent it from being used to infect others. Statistics show that most intrusions have a dwell time of weeks, months, or even years before detection. Reacting to an intrusion rapidly without following a well thought out process leads to a situation many incident responders call "whack-a-mole", as the organization blindly chases the attacker throughout the network, making little overall progress.

Whack-a-mole response occurs when you move too fast toward eradication without proper cyber threat intelligence helping direct containment and encirclement. Without good visibility and proper containment, the adversary is free to redeploy assets around the network to ensure survivability. Removal of only a portion of adversary-controlled infrastructure results in little progress and by the time your IR team is done patting itself on the back with a "job well done", the adversary will be back compromising additional systems.

What drives the immediate eradication/remediation call to arms? Many organizations fear losing the data stored on the system to the adversary. The data may be deemed as simply too valuable and the risk too high. As a result, many IR teams know it is a bad idea to remediate too early but are compelled to do so by management's fear of the horse leaving the barn. But as the analogy goes, closing the barn door after the horses have been let out is useless. Rushing to eradicate at this stage can also lead the attacker to react your remediation actions before you are prepared to counteract. They might assume your actions are the beginning of a full-scale remediation and begin a major exfiltration or destructive process on other systems they control. This act/react/counteract model is the norm for intrusion response. As a result, the better you can scope your incident and learn about your adversary, the more eventual control you will have over the results.

Bottom line: Do not react too quickly to an incident by pulling the power. Teams need to move toward intelligence-driven incident response.

Containment and Intelligence Development

Containment/ Intelligence Gathering

Intelligence Development

Tools, techniques, and procedures observation
Understanding adversary intent
Malware gathering
IOC development
Campaign identification

Containment / Active Defense

"Prevent or slow additional access during monitoring and collection phase"
Full-scale host/network monitoring
Data decoy
Bit mangling
Traffic shaping
Adversary network segmentation
"AVOID PLAYING YOUR HAND"

SANS DFIR FOR508 | Advanced Incident Response, Threat Hunting, and Digital Forensics 24

The bulk of response time is often spent in the containment and intelligence development phase of the incident response cycle. This is where responders learn about an ongoing attack and ultimately unravel the intentions of the attacker. The need for threat intelligence collection during an attack cannot be overstated. If you are not collecting information on attacker activity, you are starting from the absolute beginning in every investigation. Intelligence helps guide your network and host sweeps, facilitating rapid identification of additional compromised hosts in your environment. For example, if you find an instance of evil.exe in an unusual directory on a system, it is likely the adversary also placed a similar file in the same directory on another system. This is called an indicator of compromise, or an IOC. IOC development is extremely important at this phase because it is a force multiplier, ensuring the massive effort of finding suspicious activity can be automatically replicated across the enterprise.

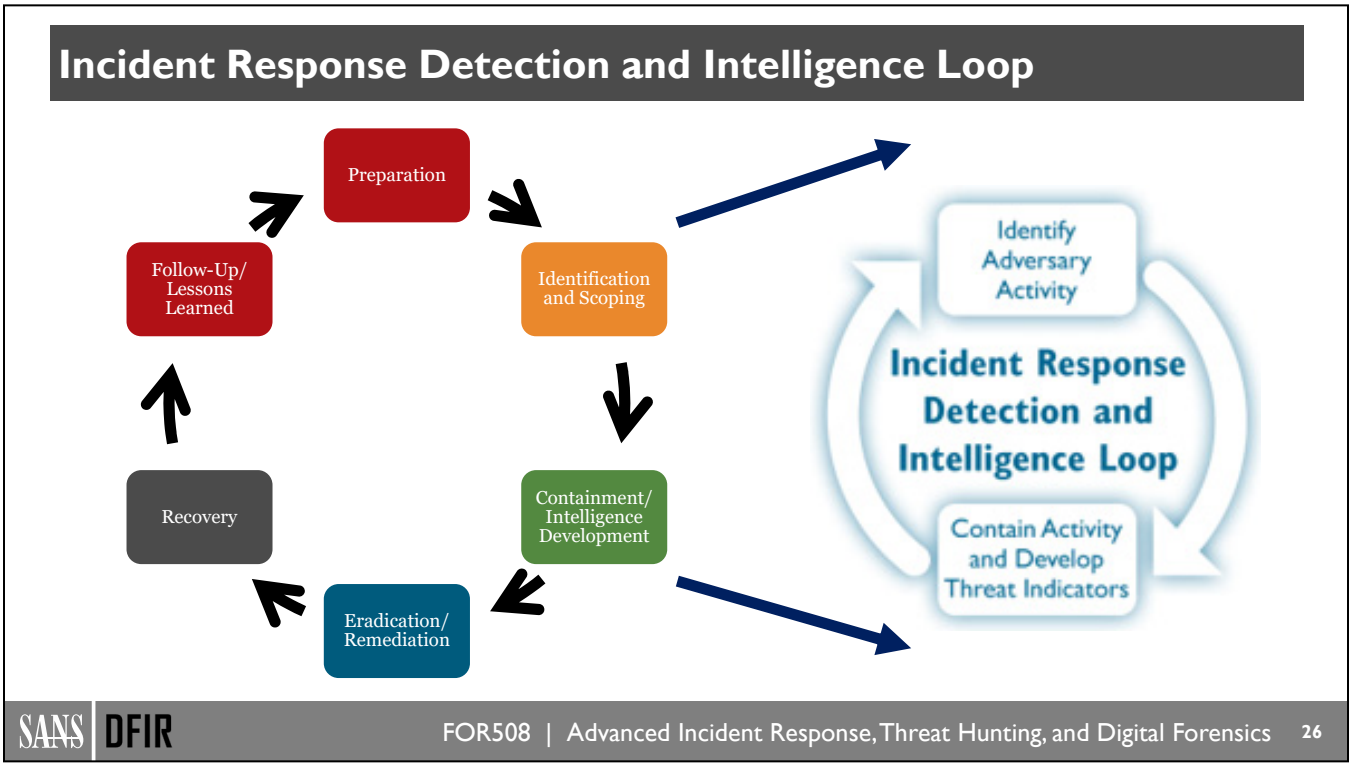
Keep in mind that an indicator does not have to be malware, and with today's advanced attackers, may be legitimate system tools and commands. A compromised host could be any system that the adversary has examined, utilized, or infected during their campaign. Eventually, with enough intelligence data, predicting an adversary's intent and future actions is possible. When this point is reached, it is time to consider moving to the eradication and remediation phase.

As the incident response team learns more about the attack and adversary, a window opens allowing containment actions to occur. As an example, intelligence collection can allow teams to predict the type of data an attacker is searching for, leading to additional security capabilities built around systems housing that type of data. The goal of containment is to degrade the capabilities of an adversary, denying them the opportunity to achieve their goals. Containment is not the end game, but it can be used sparingly to provide breathing room to the incident response team until they can enact a permanent eradication and remediation plan. During an intrusion, the more you can learn about your adversary's capabilities and true intent, the easier it is to achieve containment. For example, if the adversary simply wants to steal intellectual property and spy on your organization, limiting the adversary from exfiltrating the stolen data would prevent them from achieving operational success. Any technique you employ to restrict, limit, and degrade the capabilities of your adversary while moving around the network, collecting data, and exfiltrating would be considered part of containment.

The course authors learned back in 1998 while responding to one of the earliest nation-state attacks, “Moonlight Maze”, that anytime responders react too quickly to an intruder, the attackers will have an equal response.[1] In some cases, the Russian-based actors would go dormant for months and resurface on the same network once they felt they were not being searched for any longer. This led us to begin heavy monitoring with a kill switch. Essentially, we used a bridging firewall to segment compromised network segments and systems that the Russian APT was targeting. If they began to collect data for exfiltration or attempted something we didn’t like, we dropped the connection or temporarily blocked the network. We broadcasted many reasons for the “downtime” in case the intruder suspected they were being watched. On the flip side, we also learned attackers were monitoring the “uptime” of systems to ensure the power had not been cut to allow for disk imaging, which was an indicator to the attackers that they had been discovered. Our strategy quickly evolved from pull the plug and image the system to “image live” and “monitor with a kill switch.” Incident response is a chess match, and the better you understand your opponent, the more moves you can make on the board.

Active defense and containment capabilities employed by responders include decoy datasets, corrupting data, preventing data exfiltration or lateral movement, and employing kill switches on the network. Full-packet capture monitoring of adversary-controlled systems and network segments can be incredibly useful. In some cases, organizations have deployed their own host-based monitors and key loggers to specifically spy on the adversary’s every move. Containment is only limited by your intelligence capability, creativity, and resources.

[1] Moonlight Maze: https://for508.com/8uh5y



In practice, the two critical phases of "Identification and scoping" and "Containment / Intelligence Development" form a mini-cycle unto themselves. Information gleaned during forensic analysis in the intelligence collection phase is used to further scope the network, identifying previously unknown compromised systems. Those new systems are analyzed, providing additional information on adversary actions and new IOCs, which are then used to find even more systems. This synergistic loop continues until the incident response team believes they have fully scoped the incident and are ready to attempt eradication and remediation of the environment.

Remediation Is Hard

Eradication/ Remediation

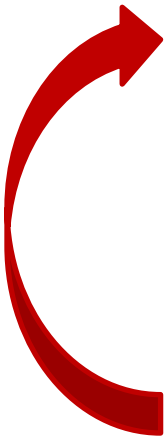
Threats are good at avoiding detection and ensuring survivability

Threats react to countermeasures and remediation tactics

Threats will return

Nothing is more important to an organization than finally removing the adversary threat from the network. Sadly, this is much easier said than done and most organizations move to this step too soon after incident detection. Without proper scoping and intelligence collection, remediation is simply not possible. Pre-emptive remediation ends up only annoying the adversary, causing them to change tactics, and ultimately making it harder for the IR team to track and complete the incident response process. In the end, remediation is a part of an ongoing incident response cycle. Many organizations fail in their first attempts at remediation. It can be very difficult to fully scope an intrusion in a large network and ensure that all attacker command and control infrastructure is accounted for. During these failures, the team will re-trench, go back to the scoping phase and continue their intelligence collection on the adversary. In this way, remediation is much more like a relay race than a sprint.

A typical failed remediation might happen like this:



1. Response team removes/rebuilds all known compromised hosts and blocks IP addresses, domains, resetting possible compromised accounts.
2. Response team, not having scoped out the full intrusion before tipping its hand, ends up showing the adversary how it found them by removing specific systems.
3. Adversary, not knowing if the response team is any good, deploys new malware to ensure long-term access to the network.
4. Response team and management feel a sense of satisfaction, as they "removed the threat" from their environment.
5. Attacker maintains access but is using new capabilities not seen before by the response team.
6. This continues until the attacker makes a mistake or an external organization notifies the organization of the intrusion, and the cycle generally repeats itself.
7. Go to step 1; full eradication did not take place. The adversary survived remediation.

Why is successful remediation so difficult to accomplish? Modern adversaries are veterans at the game. They are extremely good at avoiding detection and ultimately plan on being detected at some point. As a result, they often go to great lengths to ensure survivability beyond attempts at remediation. Even when an attacker is successfully remediated, expect a new wave of attacks immediately following. The typical adversary today is a well-resourced, professional organization with the time and resources available to continually attempt re-entry into a network.

Remediation Events

Remediation takes time to plan. Planning should start almost immediately after the start of an incident response. It almost always involves additional groups outside of the incident response team, with massive coordination required to enact a burst of network changes over a short period of time. This is commonly called a "Remediation Event." Remediation events are often planned over a weekend when an organization can commit to purging an adversary from its network without greatly impacting business operations.

A remediation event should:

1. Deny access to the environment
2. Eliminate the ability for the adversary to react to the remediation
3. Remove the presence of the adversary from the environment
4. Degrade the ability of the adversary to return

Remediation consists of three steps:

1. Posture for remediation
2. Execute remediation
3. Implement and apply additional security controls

Critical Remediation Event Steps

During the remediation, there is no one right solution to apply. This is one of the reasons that remediation planning takes some time to complete. Understanding and knowing your adversary through intelligence-driven incident response is key. You need to know every host and system compromised by your attackers. You should detail security controls that would degrade and deny the ability of the adversary to function properly. Regardless of the specific tactical options you might consider planning during remediation, here are a few critical recommendations we recommend you consider.

Critical remediation controls include but are not limited to:

1. Disconnect the environment from the Internet.
2. Implement strict network segmentation not allowing specific subnets to communicate with each other.
3. Block IP addresses and domain names for known C2 channels.
4. Remove all infected systems that maintained active or previous active malware on the host.
5. If needed, remove all systems identified as compromised but do not show signs of infection via malware.
6. Restrict access to known compromised accounts.
7. Restrict access to domain administrator accounts.
8. Validate that everything above is done properly.

The last step is incredibly important because people make mistakes and oversights. Leaving one compromised host online after a remediation event likely means you will need to start over from the beginning. Everyone wants this to succeed, so it is imperative that there are two sets of eyes to verify that each task is done properly. A mistake during this phase is costly because it not only allows continued access for the adversary, but it also tips your hand of what you know about them. The adversary is likely to immediately scramble to deploy new malware, maintain a presence by infecting additional hosts that might have already been cleaned up, and change their tactics to become more invisible to the incident response team.

Finally, once the remediation is successful, additional security controls should be deployed within the environment. You want to implement additional measures to increase the chance of detection of the adversary while degrading the ease of maneuverability to the threat. There are many new solutions that can be implemented but following the SANS Critical Controls is a good first step.[1] Most organizations underestimate how implementing the basics, like the top four critical controls, makes incident response and active defense much more doable and less costly. Eliminating noise in an environment makes finding future adversaries much easier.

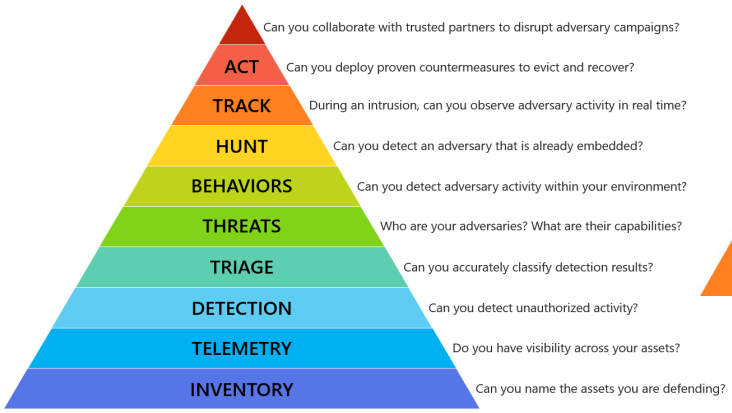
[1] http://for508.com/lqe4r

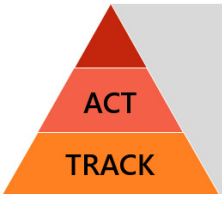
Real-Time Remediation

Eradication/ Remediation

Advances in network and endpoint monitoring provide *some* organizations with the ability to mitigate attacks in real-time

- Requires complete enterprise visibility and mature processes





"During incident response, I operate at the same tempo as the adversary to protect my business assets."

With proper visibility, remediation can (and should) begin on day one of an incident. The new way forward in incident response is a remediation-focused approach. The primary goal of any incident response is to successfully remediate the environment by eliminating attacker access and preventing further data loss. Visibility allows responders to initiate these actions much earlier in the response cycle, actively countering threats as they are found, instead of waiting until weeks or months after the initial incident to perform a large-scale sneak attack with no guarantee of success. Visibility is king, and like any battlefield, incident responders are in a race to take the high ground from the adversary. Advances in network and endpoint forensics coupled with lowered costs of storage and computational power can now provide historical visibility unheard of even a few years ago. Once an indicator of attack is discovered, the ability to go back in time and see where else that activity was recorded significantly reduces identification and containment times and greatly increases the cost to the adversary. Endpoint detection and response tools are widely providing this capability.

Of course, everything takes work, and instrumenting your network for visibility requires a proactive approach best accomplished when not in the heat of an incident. Given the near inevitability of a compromise, the most successful teams are putting the work in up front to prepare the battleground. Erecting watchtowers, testing defenses and detection capabilities, and re-architecting networks to reduce the attack surface of oft-abused credentials and egress points all pay dividends far out of proportion with the upfront effort. Investing in threat intelligence and better instrumented networks also facilitates active hunting on the network. In short, most organizations have a lot of work on the basics to accomplish before challenging adversaries in real-time. But the goal is achievable—there are organizations, at this moment, operating at the top of the pyramid at the same tempo as their adversaries.

Incident Response Hierarchy of Needs provided under Creative Commons by Matt Swann
https://github.com/swannman/ircapabilities

Reactive Response vs. Threat Hunting

Reactive Organization

- Incident starts when notification comes in
  - Call from government agency
  - Vendor/threat information
  - Security appliance alert
  - "Five-alarm fire" response

Hunting Organization

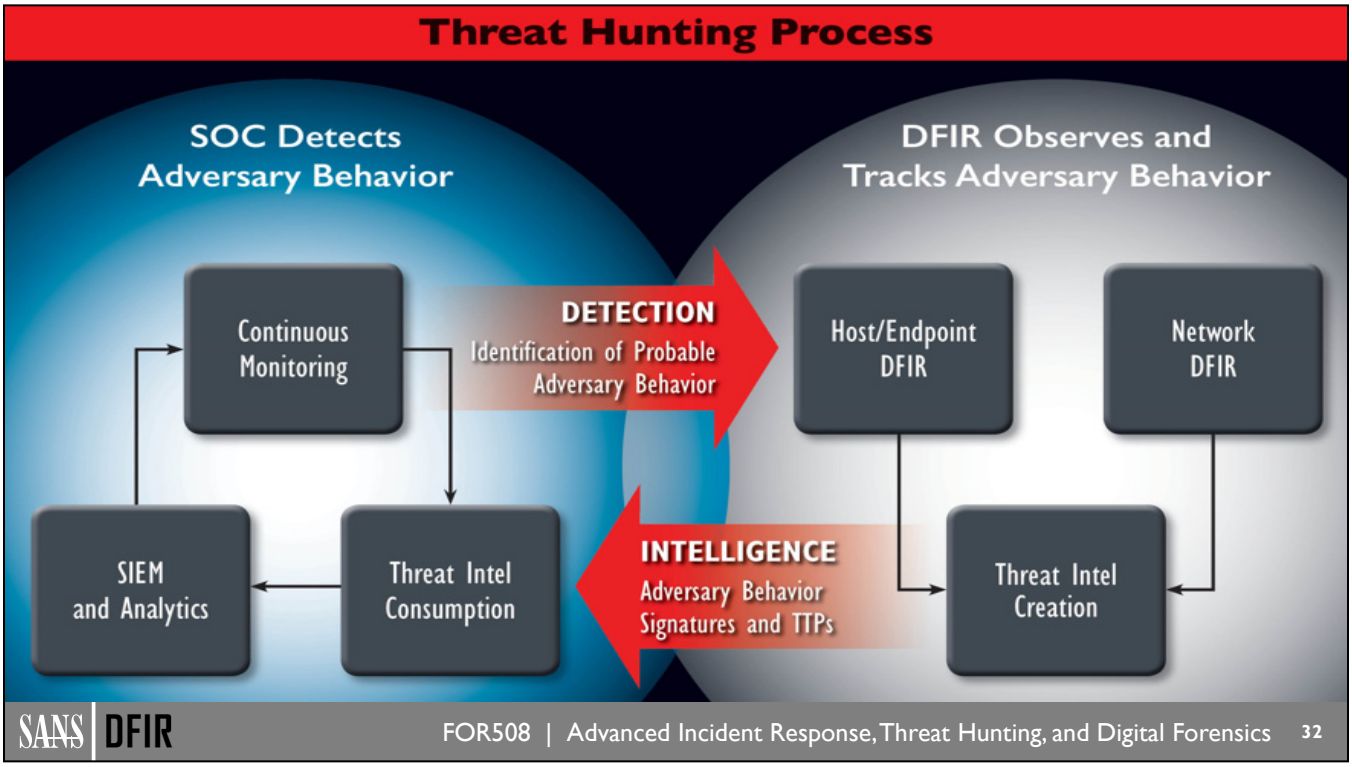
- Actively looking for incidents
  - Known malware and variants
  - Patterns of activity: evil versus normal
  - Threat intelligence
  - Security patrols
- Reduce adversary dwell time

SANS DFIR FOR508 | Advanced Incident Response, Threat Hunting, and Digital Forensics 31

A reactive organization begins incident response when an alert or notification comes in. The alert could come from a third party such as the FBI, or it could come from the organization's own security sensors. The best analogy to a reactive approach is the IR team is largely waiting to be called into action and relying on the accuracy of the notifications it is receiving. Most organizations start building their incident response teams as a reactive organization. In many cases, the IR team is comprised of augmentation staff that normally fulfill other duties during their regular jobs. As the organization grows larger or as it has an increasing number of incidents, the team might become permanent. The best analogy here is small towns with volunteer fire departments versus full-time fire fighters. Even larger organizations augment their IR teams with additional personnel internally or contract with consultants who provide surge-based incident response services.

Organizations should move to a hunting posture when they have optimized their IR process and are still not detecting incidents early enough. A primary goal of hunting is to reduce the dwell time of attackers. In this model, instead of waiting for alerts to appear, hunt team members actively scour the enterprise to identify attacks in progress. This is very similar to internal security patrols used by the military to protect bases and territory. Moving to hunting-based response is not an "either or" approach. Most hunting organizations are also reactive organizations, but they begin to task their incident response team to also actively engage and hunt for adversaries. To accomplish this task, the hunting team typically will start with known malware of interest, patterns of activity, or accurate threat intelligence aiding them in their search.

Organizations who decide to create a hunting organization sometimes fail to see the importance of proper preparation and threat intelligence for driving the search in the right areas. Simply tasking a team to "find evil" is not enough. The team needs to know the difference between normal and abnormal in the environment. It needs to know typical hacker tools and techniques. It needs to be skilled in both network and host-based forensics to look for attack residue. Finally, it helps if the organization has invested in a cyber threat intelligence capability that will accurately guide and help the team prioritize the right indicators and right locations on the network to search for the latest attacks occurring in the wild.



Threat hunting has become popular across the industry simply because it works. As a good pen test will demonstrate, attackers can be infinitely creative with finding new and undocumented ways to breach a network. The only current way to counter those types of attacks is with other humans inventing equally clever ways to detect attacker activity. Hunting, in its current state, is human vs. human with technology used as a force multiplier. Hunt teams can be a critical part of your security eco-system, discovering novel ways to detect the latest threat activity that is not already being detected with automated tools. As the graphic on this slide shows, that information should flow back to your security operations center (SOC) to be automated and folded into your continuous monitoring process. Said another way, hunt teams should be looking for new detection methods that can then be automated for enterprise-wide detection. They can also uncover data and reporting deficiencies that can pay dividends when incidents occur.

IR and Hunt Team Roles

Digital forensics is the process used to analyze systems (host and network data) in order to identify compromised systems and provide guidance on necessary remediation steps. Being able to properly collect intelligence depends greatly on the team's ability to analyze the remnants of intruder activity and use that information to identify other compromised infrastructure. As a result, your team should be made up of host, network, and reverse engineers working side by side to identify newly compromised systems, create new threat intelligence data, and use that data to identify new systems or to engineer additional defenses to help contain the current incident. A suggested team composition follows:

1 Team lead
1–2 Endpoint / host / cloud analysts
1–2 Network analysts
1 Reverse engineering malware specialist
1 DevOps / tool development resource

Humans are the most important components of a hunt team (and a security team in general), but technology is also important. The team should have an enterprise scanning capability for both host and network-based signatures. Do not fall into the trap of believing hunting requires sophisticated commercial tools. A team could easily use built-in PowerShell remoting to accomplish amazing things. In a nutshell, outfitting a hunt team requires the ability to access a wide variety of data sources and capabilities in the enterprise. The more options a team has for gathering and examining data, the higher the success rate.

Cyber Threat Intelligence Role in Hunt Teaming
A key component to building a hunt team is a cyber threat intelligence capability residing inside your security team and feeding directly to the hunt team. It is always surprising to hear organizations that have committed to staffing a hunting team and when asked what the team is looking for on the network, the response is "we don't know." A proper cyber threat intelligence capability will arm the hunting team with:
Where to look: In what data might sophisticated threat actors be particularly interested?
What to look for: Host, network, and malware artifacts, registry keys, tools such as PSEXEC, living off the land binaries, WMI, PowerShell, etc.
Likelihood of attack: Which threat groups are most likely to target our network?

Right Mindset
Hunt teams require both manual and automated methods of collecting and searching data across an enterprise network. A single hunt team member should be able to scale up to searching thousands of hosts for a single forensic artifact. Alternatively, they might need to dive deeply into a single system trying to uncover unknown malware. The challenge is always to know when analysis is complete, and the hunter should move on. The analyst will always feel like they missed something, did not have the right skills to find it, or the adversary is simply better than them. No amount of searching will help remove the doubt that comes with hunting and not finding anything of substance. A good hunt team manager will constantly need to nudge analysts on to the next artifact to look for.

Without any type of threat intelligence, most hunting groups are simply tasked with looking for "things that look weird" perhaps without even knowing what weird versus normal looks like. A trained hunter must know the difference between normal and abnormal as a prerequisite. Even better, if a threat intelligence capability is informing the team, they will organize hunts to look for specific threat groups targeting specific programs using specific techniques. This is an achievable goal.

Hunt Team Operational Tempo
A common challenge of hunt teams is operational tempo. Incident response initiated by reactive response teams is usually a sprint with long days, seven days a week, until the incident is remediated. While this scenario is typical for reactive response teams it could be the death of your hunt team.

Hunt teams will consistently find new breaches if they are good. If every breach detected is treated like a 24/7 fire drill, the team will soon be exhausted. Fire fighters fighting forest blazes for week take days off to recuperate. In fact, the operational tempo is purposely slower to ensure no one is working a fire line who is exhausted. Forcing your incident response hunt teams to take days off, weekends off, vacations, and spending time with family is a must. They should come in at 8 AM and work till 5 PM like everyone else. The only difference is that a hunt team works its nine hours a day finding evil and responding to it consistently. Good hunt teams will always be in the middle of an engagement fighting an adversary—fighting the adversary is simply what their job is. It is not a fire drill.

Management Support of the Hunt Team

Management buy-in is also a must. In many cases, management thinks it wants to know about breaches, but we have found over the years many organizations are far more concerned about what happens when breaches are found rather than whether they exist in the first place. This could be due to breach reporting requirements or regulatory fines levied if a breach is discovered. You might get management buy-in on paper, but management may see hunt teaming like going to the doctor to see whether they have contracted a superbug disease or cancer. Few people really want the truth even when it is good for them in the long run. Once management knows about an intrusion, they must do something about it.

Now, not all management ignores the usefulness of a hunt team. Generally, organizations that are hit by enough adversaries begin to form a thickness of skin and desensitization to the news of a breach. Strong management will simply want to know how effective the team is and how many adversaries it is currently tracking. In our experience, management warms up to the idea of hunt teams eventually, but do not be surprised if they are not excited initially when the hunt team ends up being a little too successful.

This page intentionally left blank.

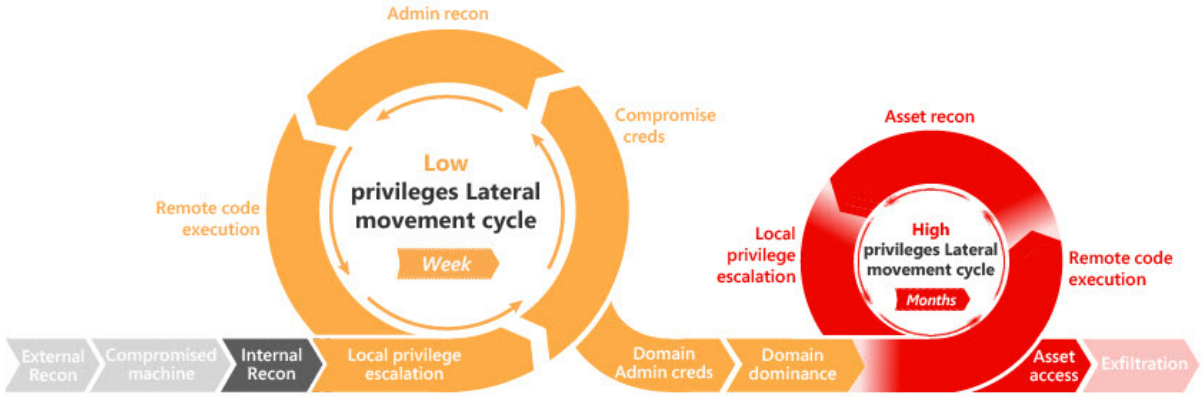
Threat Intelligence

Tracking Attacker Behavior

This page intentionally left blank.

The Attack Lifecycle

Threat intelligence maps attacker techniques, tactics, and procedures (TTPs) to the attack lifecycle

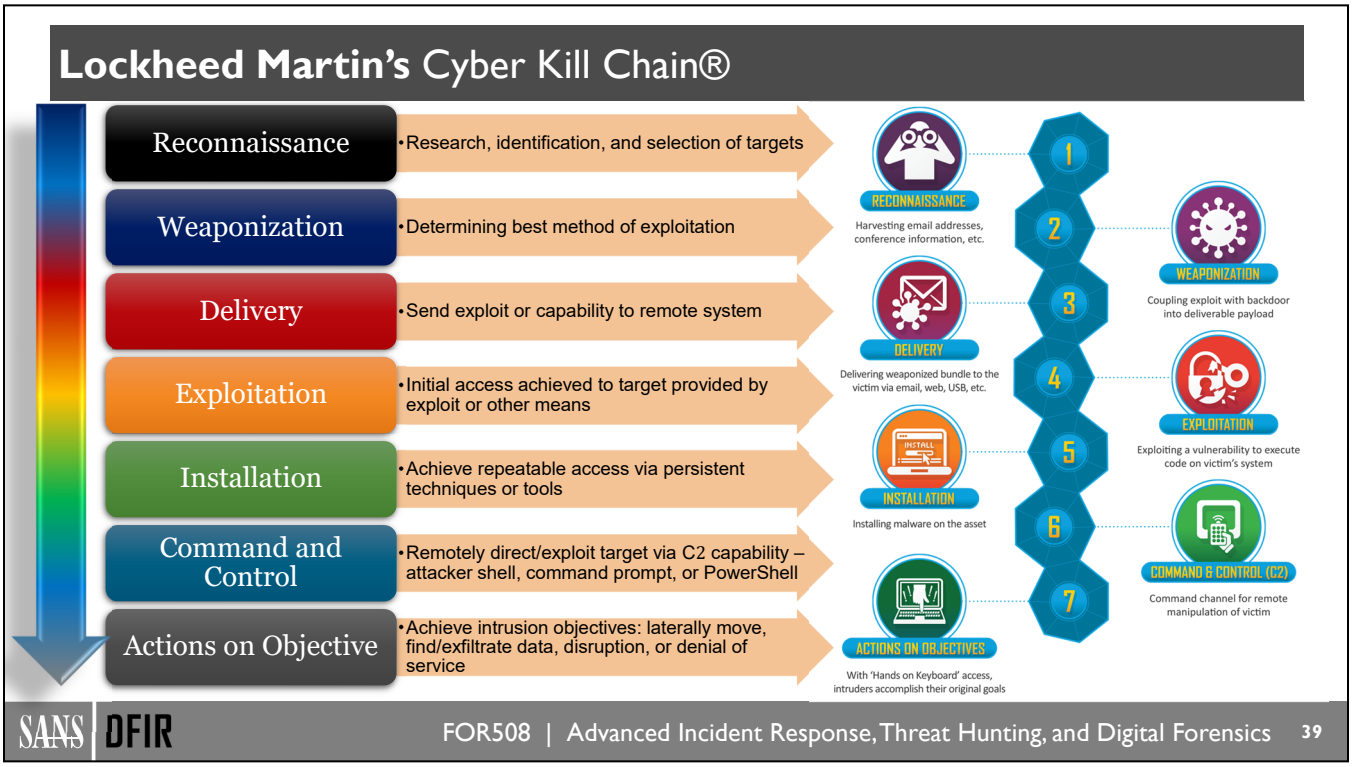


Threat intelligence attempts to map attacker techniques, tactics, and procedures to the attack lifecycle. Nearly all attacks progress through a series of common steps to accomplish adversary objectives. Understanding these steps allows defenders to craft opportunities to discover attacks while in progress. The model on this slide is sourced from documentation on Microsoft Advanced Threat Analytics, and it does a commendable job of demonstrating the cyclical nature of attacks as attackers perform network discovery and elevate privileges to achieve greater access.

- **Initial compromise:** Most initial compromises are not persistent, and the level of access achieved by an adversary at this stage is generally very fragile. If a response team can eliminate an adversary before they establish a foothold on the network, then survivability of the adversary drops to nearly zero. Unfortunately, this can be extremely hard to accomplish because initial reconnaissance and exploit delivery leave few discernable artifacts that rise above the noise of constant attacks against the network.
- **Low privileges lateral movement cycle:** During this phase an attacker must maintain persistence within the environment and expand their foothold, gaining access to additional systems. The placement of persistent backdoors is common. Large amounts of lateral movement paired with credential dumping is seen during this phase as high-level credentials are required to achieve most attacker goals.
- **High privileges lateral movement cycle:** Once high-level credentials are achieved, the attack shifts from mass credential collection to asset collection as the adversary extends their influence within the environment and prepares to accomplish their ultimate objectives. Since high-level credentials are (sadly) often discovered early in the attack, the phase can comprise the longest length of time of the intrusion. Tracking unusual credential usage, lateral movement, and abnormal system access are important capabilities in finding attacker activity in this phase.

- **Asset access and data exfiltration:** Adversaries leave many footprints as they search for and collect data of interest. The process of finding specific data on a remote network is challenging. You might have a hard time doing it on your own systems; imagine what it would be like to try and find an important file out of millions of files across thousands of hosts! Once assets and data are identified, an attacker must find a way to exfiltrate it. This is often easier said than done because moving a large quantity of data outside the network is likely to be caught by network monitoring. It is common for attackers to find and utilize a "staging system" in order to accomplish this goal. An alternative to data exfiltration is the setup of destructive attacks like ransomware. Similar to data exfiltration, the most sophisticated attackers can spend weeks or even months learning about a network before deploying their final payloads.

The flip side of this lifecycle is the incident response team charged with identifying intrusion activity. The best teams employ intelligence-driven incident response, using the identification/scoping and containment/intelligence stages of the incident response cycle to understand an adversary both during and after an incident. By collecting intelligence at each phase of the attack lifecycle, defenders can more easily counter adversary activity and employ mechanisms, both automated and manual, to find them in the future. The latter point is important as it implies the same adversary will likely be seen again, and the knowledge gained during this intrusion can allow faster mitigation of future attacks.



The "Kill Chain" is an important model used in threat intelligence. It helps categorize the sequence of actions occurring in most attacks and provides a framework for organizing detection indicators. It is best described by one of its authors, Mike Cloppert:

Security Intelligence: Attacking the Cyber Kill Chain
Now we will introduce the attack progression (known as "kill chain") and briefly describe its intersection with indicators. The next segment will go into more detail about how to use the attack progression model for more effective analysis and defense, including a few contrived examples based on real attacks.

On Indicators
Just like everyone, adversaries have various computer resources at their disposal. They have favorite computers, applications, techniques, websites, etc. It is these fundamental human tendencies and technical limitations that we exploit by collecting information on our adversaries. No person acts truly random, and no person has truly infinite resources at their disposal. Thus, it behooves us in CND to record, track, and group information on our sophisticated adversaries to develop profiles. With these profiles, we can draw inferences, and with those inferences, we can be more adaptive and effectively defend our data. After all, that's what intelligence-driven response is all about: defending data that sophisticated adversaries want. It's not about the computers. It's not about the networks. It's about the data. We have it, and they want it.

Indicators can be classified a number of ways. Over the years, my colleagues and I have wrestled with the most effective way to break them down. Currently, I am of the mind that indicators fall into one of three types: atomic, computed, and behavioral (or TTPs).

Atomic indicators are pieces of data that are indicators of adversary activity on their own. Examples include IP addresses, email addresses, a static string in a Covert Command and control (C2) channel, or fully qualified domain names (FQDNs). Atomic indicators can be problematic because they might or might not exclusively represent activity by an adversary. For instance, an IP address from where an attack is launched could very likely be an otherwise legitimate site. Atomic indicators often need vetting through analysis of available historical data to determine whether they exclusively represent hostile intent.

Computed indicators are those that are, well, computed. The most common among these indicators are hashes of malicious files, but they can also include specific data in decoded custom C2 protocols, etc. Your more complicated IDS signatures might fall into this category.

Behavioral indicators are those that combine other indicators (including other behaviors) to form a profile. Here is an example: Bad guy 1 likes to use IP addresses in West Hackistan to relay email through East Hackistan and target our sales folks with trojaned MS Word documents that discuss our upcoming benefits enrollment, which drops backdoors that communicate to A.B.C.D. Here, we see a combination of computed indicators (geolocation of IP addresses, MS Word attachments determined by a magic number, and base64 encoded in email attachments), behaviors (targets sales force), and atomic indicators (A.B.C.D C2). Already, you can probably see where we're going with intelligence-driven response. What if we can detect, or at least investigate, behavior that matches that which I described previously?

One likes to think of indicators as conceptually straightforward, but the truth is that proper classification and storage has been elusive. I'll save the intricacies of indicator difficulties for a later discussion.

Adversary Behavior

The behavioral aspect of indicators deserves its own section. Indeed, most of what we discuss in this installment centers on understanding *behavior*. The best way to behaviorally describe an adversary is by how they do their job. After all, this is the only discoverable part for an organization that is strictly CND (some of our friends in the USG likely have better ways of understanding adversaries). That "job" is compromising data, and therefore we describe our attacker in terms of the anatomy of her attacks.

Ideally, if we could attach a human being to each and every observed activity on our network and hosts, we could easily identify our attackers and respond appropriately every time. At this point in history, that sort of capability passes beyond pipe dream into ludicrous. However mad this goal is, it provides a target for our analysis: We need to push our detection "closer" to the adversary. If all we know is the forged email address an adversary tends to use in delivering hostile email, assuming this is uniquely linked to malicious behavior, we have a mutable and temporal indicator upon which to detect. Sure, we can easily discover when it's used in the future, and we are obliged to do so as part of our due diligence. The problem is this can be changed at any time on a whim. If, however, the adversary has found an open mail relay that no one else uses, then we have found an indicator "closer" to the adversary. It's much more difficult (though, in the scheme of things, still somewhat easy) to find a new open mail relay to use than it is to change the forged sending address. Thus, we have pushed our detection "closer" to the adversary. Atomic, computed, and behavioral indicators can describe more or less mutable/temporal indicators in a hierarchy. We as analysts seek the most static of all indicators, at the top of this list, but often must settle for indicators further from the adversary until those key elements reveal themselves.

That this analysis begins with the adversary and then dovetails into defense makes it very much a security intelligence technique, as we've defined the term. Following a sophisticated actor over time is analogous to watching someone's shadow. Many factors influence what you see, such as the time of day, angle of the sun, etc. After you account for these variables, you begin to notice nuances in how the person moves, observations that make the shadow distinct from others. Eventually, you know so much about how the person moves that you can pick him out of a crowd of shadows. However, you never know for sure if you're looking at the same person. At that point, for our purposes, it doesn't matter. If it looks like a duck and sounds like a duck... it hacks like a duck. Whether the same person (or even group) is truly at the other end of behavior every time is immaterial if the profile you build facilitates predicting future activity and detecting it.

Attack Progression, or the Kill Chain

We have found that the phases of an attack can be described by six sequential stages. Once again, loosely borrowing vernacular, the phases of an operation can be described as a “kill chain.” This is a linear flow—some phases might occur in parallel, and the order of earlier phases can be interchanged—but rather how far along an adversary has progressed in his or her attack, the corresponding damage, and investigation that must be performed.

Recon
The reconnaissance phase is straightforward. However, in security intelligence, oftentimes this is manifested not in portscans, system enumeration, or the like. It is the data equivalent: browsing websites, pulling down PDFs, learning the internal structure of the target organization. A few years ago, I never would've believed that people went to this level of effort to target an organization, but after witnessing it happen, I can say with confidence that it does. The problem with activity in this phase is that it is often indistinguishable from normal activity. There are precious few cases where one can collect information here and find associated behavior in the delivery phase matching an adversary's behavioral profile with high confidence and a low false positive rate. These cases are truly gems; when they can be identified, they link what are often two normal-looking events in a way that greatly enhances detection. The weaponization phase might or might not happen after reconnaissance; it is placed here merely for convenience. This is the one phase that the victim doesn't see happen but can very much detect. Weaponization is the act of placing malicious payload into a delivery vehicle. It's the difference in how a Russian warhead is wired to the detonator versus how a U.S. warhead is wired in. For us, it is the technique used to obfuscate shellcode, the way an executable is packed into a trojaned document, etc. Detection of this is not always possible, nor is it always predictable, but when it can be done, it is a highly effective technique. Only by reverse engineering of delivered payloads is an understanding of an adversary's weaponization achieved. This is distinctly separate and often persistent across the subsequent stages.

Delivery
Delivery is rather straightforward. Whether it is an HTTP request containing SQL injection code or an email with a hyperlink to a compromised website, this is the critical phase where the payload is delivered to its target. I heard a term just the other day that I really like: "warheads on foreheads" (courtesy of the U.S. Army).

Exploitation
The compromise phase will possibly have elements of a software vulnerability, a human vulnerability known as "social engineering," or a hardware vulnerability. Although the latter are quite rare by comparison, I include hardware vulnerabilities for the sake of completeness. The compromise of the target might itself be multiphase or more straightforward. As a result, we sometimes have the tendency to pull apart this phase into separate sub-phases or peel out "Compromise" and "Exploit" as wholly separate. For simplicity's sake, we'll keep this as a single phase. A single-phase exploit results in the compromised host behaving according to the attacker's wishes directly as a result of the successful execution of the delivered payload, for example, if an attacker coaxes a user into running an EXE attachment to an email, which contained the desired backdoor code. A multiphase exploit typically will involve delivery of shellcode whose sole function is to pull down and execute more capable code upon execution. Shellcode often needs to be portable for a variety of reasons, necessitating such an approach. We have seen other cases where, possibly through sheer laziness, adversaries end up delivering exploits whose downloaders download other downloaders before finally installing the desired code. As you can imagine, the more phases involved, the lower an adversary's probability of success.

This is the pivotal phase of the attack. If this phase completes successfully, what we as security analysts have classically called "incident response" is initiated: Code is present on a machine that should not be there. However, as will be discussed later, the notion of "incident response" is so different in intelligence-driven response (and the classic model so inapplicable) that we have started to move away from using the term altogether. The better term for security intelligence is "compromise response" because it removes ambiguity from the term "incident."

C2: Maintain Presence
The command and control phase of the attack represents the period after which adversaries leverage the exploit of a system. A compromise does not necessarily mean C2, just as C2 doesn't necessarily mean exfiltration. In fact, we will discuss how this can be exploited in CND but recognize that successful

communications back to the adversary *often* must be made before any potential for impact to data can be realized. This can be prevented intentionally by identifying C2 in unsuccessful past attacks by the same adversary, resulting in network mitigations, or fortuitously when adversaries drop malware that is somehow incompatible with your network infrastructure, to give but two examples.

In addition to the phone call going through, someone has to be present at the other end to receive it. Your adversaries take time off too... but not all of them. In fact, a few groups have been observed to be so responsive that it suggests a mature organization with shifts and procedures behind the attack more refined than that of many incident response organizations.

Actions on Objectives

The Actions on Objectives phase is conceptually very simple: For most APTs, this is when the data, which has been the ultimate target all along, is taken. Previously, I mentioned that gathering information about the environment of the compromised machine doesn't fall into the exfiltration phase. The reason for this is that such data is being gathered to serve but one purpose, either immediately or longer term to facilitate collection and theft of the target information: the source code for the new O/S, the new widget that cost billions to develop, and access to the credit cards or PII. Adversary objectives could also fall into the category of denial of service, data destruction, or more.

We will also lump lateral movement with compromised credentials, file system enumeration, and additional tool dropping by adversaries broadly into this phase of the attack. Although an argument can be made that situational awareness of the compromised environment is, technically, "exfiltration."

Security Intelligence Using the Kill Chain Successfully

(Included with permission from Mike Cloppert as originally published on the SANS Computer Forensics Blog.)

The "persistence" in APT intrusions is manifested in two ways: maintaining a presence on your network, as well as repeatedly attempting to gain entry to areas where presence is not established. The repeatability of these activities inevitably involves attributes that are consistent because resource constraints typically prevent adversaries from acting differently every time, they set foot in your environment. With a way to model intrusions and align these common attributes, network defenders can take advantage of persistence to profile their adversaries, informing strategic response, analysis efforts, and resource investment.

A single intrusion, as we have already discussed, can be modeled as seven phases. Within each of these phases of an intrusion is a highly dimensional set of indicators—computer scientists would call them "attributes"—that together uniquely define that intrusion. For example, a C2 callback domain is an indicator attribute; talktome.bad.com is the corresponding value of the indicator. The targeting used (reconnaissance), the way in which the malicious payload is obscured (weaponization), the path the payload takes (delivery), the way the payload is invoked (exploit), where the backdoor is hidden on the system (installation), the protocol used to call back to the adversary (C2), and habits of the adversary once control is established (actions on intent) are all categorical examples of these indicators. It is up to the analyst to discover the significant or uniquely identifying indicators in an intrusion. In some cases, there are common indicators—for example, the last-hop email relay used to deliver a message will be significant in most like intrusions, excluding webmail. In others, the attributes can be unique and surprising—a piece of metadata, a string in the binary of a backdoor, and a predictably malformed HTTP request to check for connectivity.

Be aware that adversaries shift tactics over time. A campaign is not static, nor are the key indicators or their corresponding values. We've seen adversaries use the same delivery and C2 infrastructure for years, whereas others will shift from consistent infrastructure in the Delivery and C2 phases to highly variable infrastructure in

the delivery phase but consistent targeting and weaponization techniques. Some adversaries will have consistent key indicators, such as tool artifacts in the Delivery and Weaponization phases, but the specific indicator values might change over time. Without constant and complete analysis of sophisticated intrusions, knowledge of campaigns becomes stale and ineffective at predicting future intrusions.

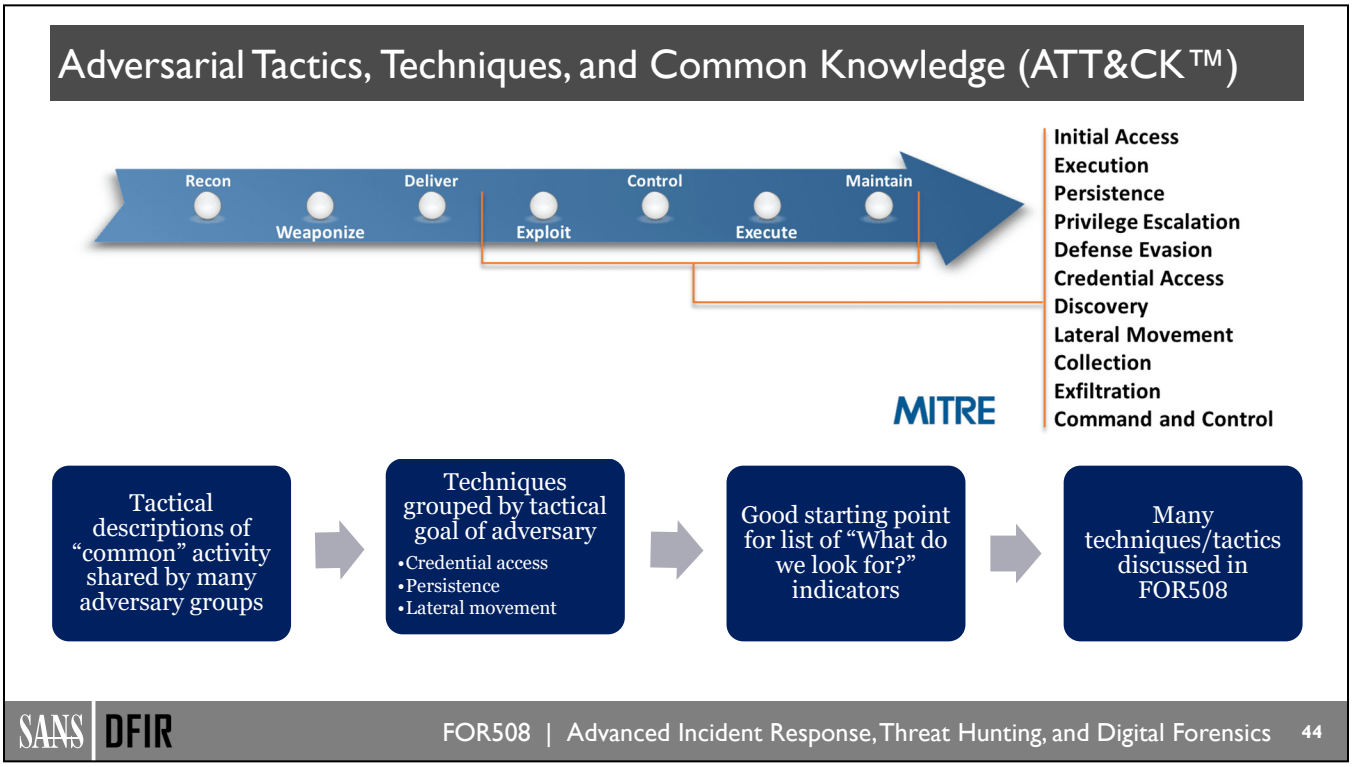
Gathering Intel through Kill Chain Completion
To have the dataset necessary to link intrusions and identify key indicators, analysts must understand all phases of every sophisticated intrusion. Initial detection of an intrusion might occur at any point across the kill chain. Even if the attack is unsuccessful, detection is just the first step.

Classic incident response methodology assumes a system compromise. In this situation, where a detection happens after the installation and/or execution of malicious code, adversaries have successfully executed many steps in their intrusion. As the intrusion progresses forward in the kill chain, so the corresponding analysis progresses backward. Analysts must reconstruct every prior stage, necessitating not only the proper tools and infrastructure to do so but also deep network and host forensic skills. Less mature response teams will often get stuck in the delivery to installation phases. Without knowledge of what happened earlier in an intrusion, network defenders will be unable to define campaigns at these earlier phases, and response to intrusions will continue to happen post-compromise because this is where the detections and mitigations are. When walls are hit in analysis that prevents reconstruction of the entire chain, these barriers represent areas for improvement in instrumentation or analytical techniques. Where tools do not already exist for accurate and timely reconstruction, development opportunities exist. Here is but one area where having developers on staff to support incident responders is critical to the success of the organization.

As response organizations mature and are able to more fully build profiles of intrusion campaigns against them, they become more successful at detection prior to compromise. However, just as a post-compromise response involves a significant amount of analysis, the unsuccessful intrusion attempts matching APT campaign characteristics also require investigation. The phases executed successfully by the adversary must still be reconstructed, and the phases that were not must be synthesized to the best ability of the responders. This aspect is critical to identifying any TTP change that may have resulted from a successful compromise. Perhaps the most attention-grabbing example is the identification of zero-day exploits used by an APT actor at the Delivery phase before the exploit is invoked.

Synthesis clearly demonstrates the criticality of malware reverse engineering skills. It is likely that the backdoor that would have been dropped, even if it is of a known family, using a known C2 protocol, also contains new indicators further defining the infrastructure at the disposal of adversaries. Examples include indicators such as C2 callback IP addresses and fully qualified domain names. Perhaps minor changes in the malicious code would produce new unique hashes, or a minor version difference results in a slightly different installation filename that could be unique. Although antivirus is typically a bad example of detection in the context of APT intrusions, there are times when it can be of value for older variants of code. For instance, how many reading this analyze emails that are detected by their perimeter antivirus system? If the detection is for a particular backdoor uniquely linked to an APT campaign, the email could contain valuable indicators about the adversary's delivery or C2 infrastructure that might be reused later in an intrusion that your antivirus system does not detect. Detecting campaigns enables resilient detection and prevention mechanisms across an intrusion and engages CND responders earlier in the kill chain, reducing the number of successful intrusions. It should be obvious but bears repeating that a lack of specific indicators from a single intrusion prevents identification of key indicators from sequential intrusions. A lack of key indicators results in an inability to define adversaries, and an inability to define adversaries leaves network defenders responding post-compromise to every intrusion. In short, inability to reconstruct intrusions should be considered an organizational failure of CND, and intelligence-based detections prior to system compromise a success. Defining campaigns, as demonstrated here, is one effective way to facilitate success.

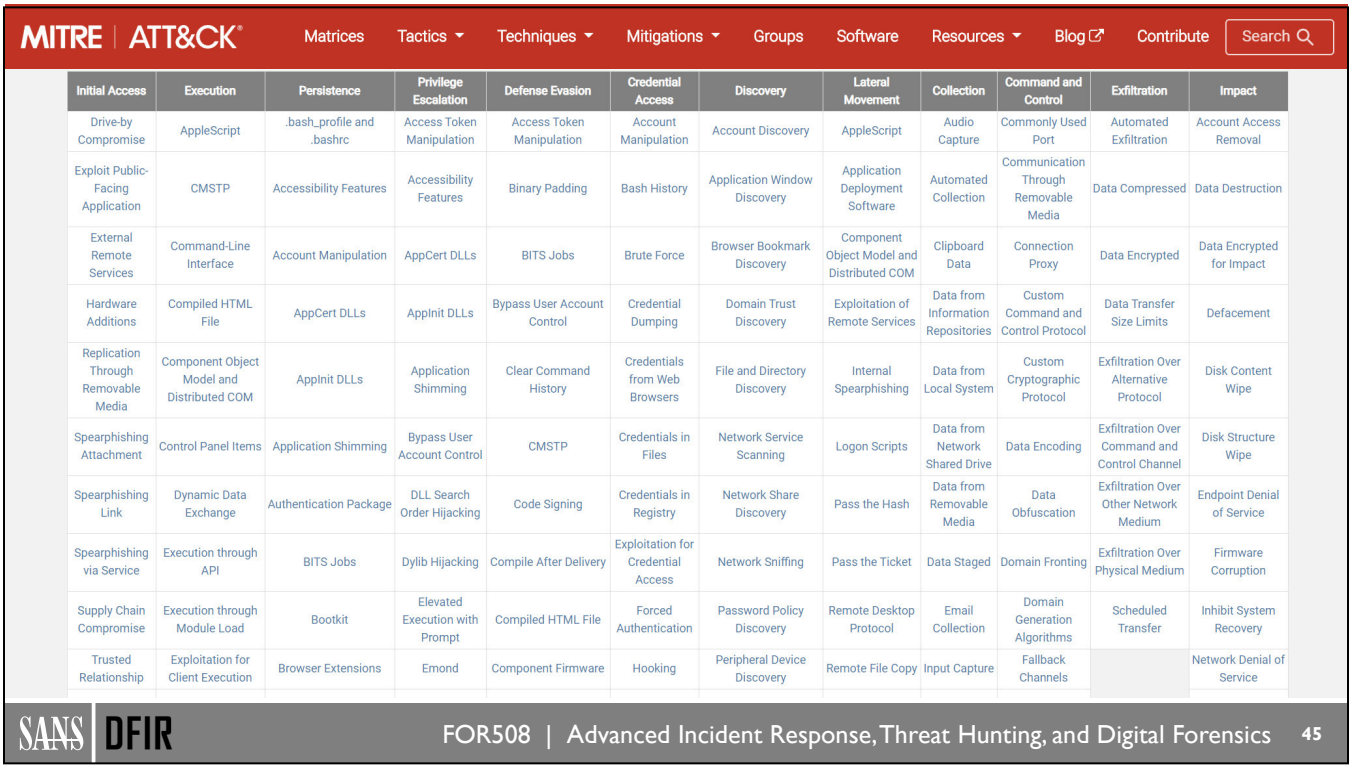
Sourced with permission from Mike Cloppert and originally published on the SANS Computer Forensics Blog at https://for508.com/1yvt6 and officially branded by Lockheed Martin at https://for508.com/s6e18



Threat hunting is often frustrating and ineffective if your team lacks effective threat intelligence. Actionable threat intelligence is key to helping your organization detect and defend against advanced attacks. Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK™) is a model and framework for describing the actions an adversary may take while operating within an enterprise network. The model is designed to help characterize and describe post-compromise adversary behavior. It both expands the knowledge of network defenders and assists in prioritizing network defense by detailing the post-compromise (post-exploit and successful access) tactics, techniques, and procedures (TTPs) advanced persistent threats (APT) use to execute their objectives while operating inside a network. Data has been compiled through collaboration with a wide range of partners and via multiple disciplines including network defense, penetration testing, and Red Teaming.

We like ATT&CK because it is currently the most detailed resource available on the universe of attacker techniques and guidance on how to hunt for evidence of them. It is threat and vendor-agnostic and focuses on TTPs all adversaries use to make decisions, expand access, and execute their objectives. It aims to describe an adversary's steps at a high enough level to be applied widely across platforms, but still maintains enough details to be technically useful. It is an excellent starting (and ending) place when building threat hunting and defensive capabilities. We frequently leverage the ATT&CK framework to ensure we cover the most relevant tactics and techniques in this class. As an example, there are a small number of lateral movement techniques shared across most threat actors, providing an obvious and lucrative place to search for evidence of compromise. You will see those techniques in an upcoming section.

ATT&CK details were sourced from the MITRE website: **https://attack.mitre.org.** ATT&CK and ATT&CK Matrix are trademarks of The MITRE Corporation



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Commonly Used Port	Automated Exfiltration	Account Access Removal
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Communication Through Removable Media	Data Compressed	Data Destruction
External Remote Services	Command-Line Interface	Account Manipulation	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Component Object Model and Distributed COM	Clipboard Data	Connection Proxy	Data Encrypted	Data Encrypted for Impact
Hardware Additions	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	Domain Trust Discovery	Exploitation of Remote Services	Data from Information Repositories	Custom Command and Control Protocol	Data Transfer Size Limits	Defacement
Replication Through Removable Media	Component Object Model and Distributed COM	AppInit DLLs	Application Shimming	Clear Command History	Credentials from Web Browsers	File and Directory Discovery	Internal Spearphishing	Data from Local System	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Content Wipe
Spearphishing Attachment	Control Panel Items	Application Shimming	Bypass User Account Control	CMSTP	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Network Shared Drive	Data Encoding	Exfiltration Over Command and Control Channel	Disk Structure Wipe
Spearphishing Link	Dynamic Data Exchange	Authentication Package	DLL Search Order Hijacking	Code Signing	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Removable Media	Data Obfuscation	Exfiltration Over Other Network Medium	Endpoint Denial of Service
Spearphishing via Service	Execution through API	BITS Jobs	Dylib Hijacking	Compile After Delivery	Exploitation for Credential Access	Network Sniffing	Pass the Ticket	Data Staged	Domain Fronting	Exfiltration Over Physical Medium	Firmware Corruption
Supply Chain Compromise	Execution through Module Load	Bootkit	Elevated Execution with Prompt	Compiled HTML File	Forced Authentication	Password Policy Discovery	Remote Desktop Protocol	Email Collection	Domain Generation Algorithms	Scheduled Transfer	Inhibit System Recovery
Trusted Relationship	Exploitation for Client Execution	Browser Extensions	Emond	Component Firmware	Hooking	Peripheral Device Discovery	Remote File Copy	Input Capture	Fallback Channels		Network Denial of Service

The twelve tactic categories for ATT&CK (ref: https://attack.mitre.org) were derived from the later stages (control, maintain, and execute) of a seven-stage Cyber Attack Lifecycle (first articulated by Lockheed Martin as the Cyber Kill Chain®).[1] This provides a deeper level of granularity in describing what can occur during an intrusion after an adversary has acquired access.

Tactics, such as "Persistence", specify common actions occurring during an attack. Each category contains a list of techniques that an adversary could use to perform that tactic. Techniques are broken down to provide a technical description, indicators, useful defensive sensor data, detection analytics, and potential mitigations. Applying intrusion data to the model helps focus defense on the commonly used techniques across groups of activity and helps identify gaps in security. Defenders and decision makers can use the information in ATT&CK for various purposes, not just as a checklist of specific adversarial techniques. Many of these techniques are covered in FOR508, but they also touch on techniques covered in FOR572 (Network Forensics) and FOR610 (Malware Analysis). In this class, we focus on techniques seen most frequently in the wild and with the highest effort-to-hit ratios, with the goal of rapid detection of intrusion activity.

ATT&CK is comprised of multiple threat matrices, documenting threats for Windows, macOS, Linux, mobile, and even cloud. This class will stay focused on the Windows Enterprise matrix. Techniques, tactics, and mitigations are covered for each element. ATT&CK is also expressed in STIX 2.0 format and can be found on the MITRE Cyber Threat Intelligence GitHub repository: **http://for508.com/ik5dy**

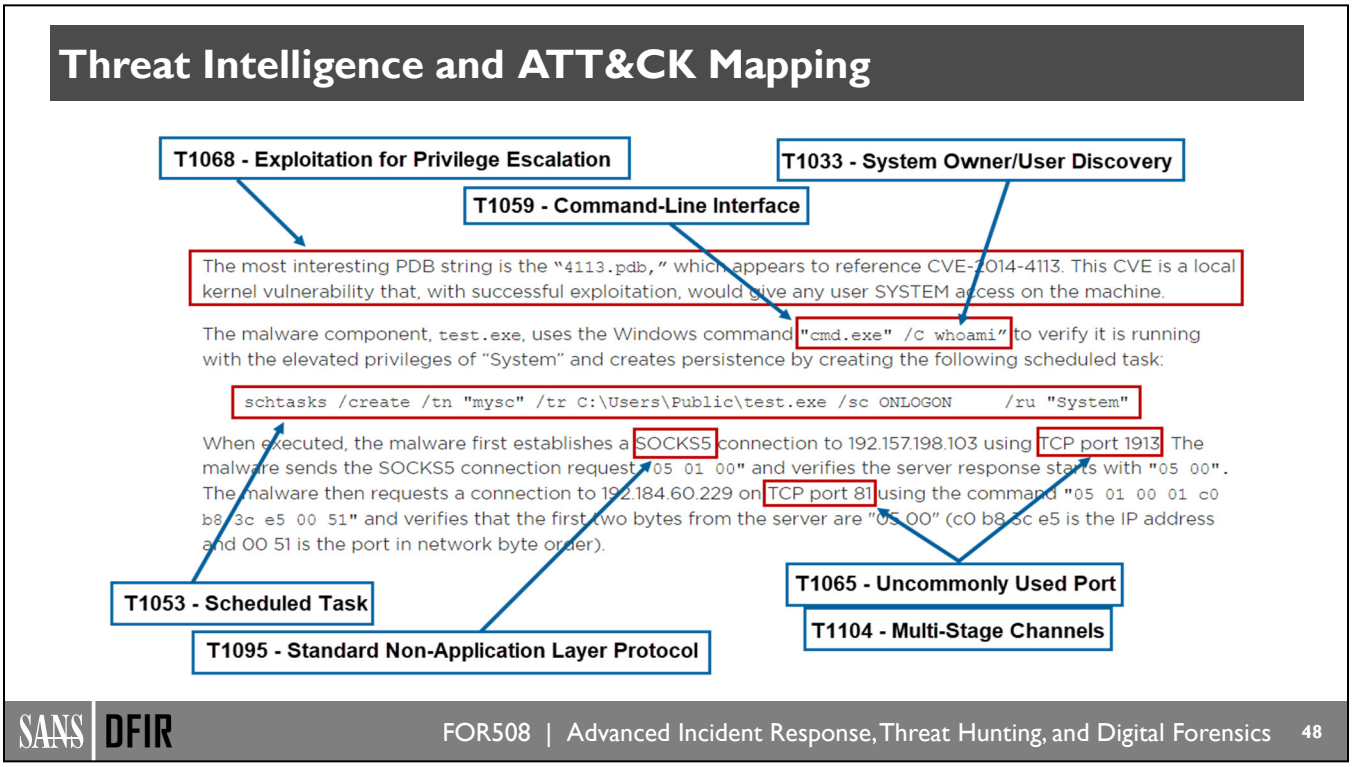
Categories covered by the Windows Enterprise ATT&CK matrix → https://attack.mitre.org

Initial Access: The adversary is trying to get into your network.
Execution: The adversary is trying to run malicious code.
Persistence: The adversary is trying to maintain their foothold.
Privilege Escalation: The adversary is trying to gain higher-level permissions.
Defense Evasion: The adversary is trying to avoid being detected.
Credential Access: The adversary is trying to steal account names and passwords.
Discovery: The adversary is trying to figure out your environment.
Lateral Movement: The adversary is trying to move through your environment.
Collection: The adversary is trying to gather data of interest to their goal.
Command and Control: The adversary is trying to communicate with compromised systems to control them.
Exfiltration: The adversary is trying to steal data.
Impact: The adversary is trying to manipulate, interrupt, or destroy your systems and data.

[1] Cyber Kill Chain: https://for508.com/kq9gl

MITRE | ATT&CK® Matrices Tactics Techniques Mitigations Groups Software Resources Blog Contribute Search

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Commonly Used Port	Automated Exfiltration	Account Access Removal
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Communication Through Removable Media	Data Compressed	Data Destruction
External Remote Services	Command-Line Interface	Account Manipulation	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Component Object Model and Distributed COM	Clipboard Data	Connection Proxy	Data Encrypted	Data Encrypted for Impact
Hardware Additions	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	Domain Trust Discovery	Exploitation of Remote Services	Data from Information Repositories	Custom Command and Control Protocol	Data Transfer Size Limits	Defacement
Replication Through Removable Media	Component Object Model and Distributed COM	AppInit DLLs	Application Shimming	Clear Command History	Credentials from Web Browsers	File and Directory Discovery	Internal Spearphishing	Data from Local System	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Content Wipe
Spearphishing Attachment	Control Panel Items	Application Shimming	Bypass User Account Control	CMSTP	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Network Shared Drive	Data Encoding	Exfiltration Over Command and Control Channel	Disk Structure Wipe
Spearphishing Link	Dynamic Data Exchange	Authentication Package	DLL Search Order Hijacking	Code Signing	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Removable Media	Data Obfuscation	Exfiltration Over Other Network Medium	Endpoint Denial of Service
Spearphishing via Service	Execution through API	BITS Jobs	Dylib Hijacking	Compile After Delivery	Exploitation for Credential Access	Network Sniffing	Pass the Ticket	Data Staged	Domain Fronting	Exfiltration Over Physical Medium	Firmware Corruption
Supply Chain Compromise	Execution through Module Load	Bootkit	Elevated Execution with Prompt	Compiled HTML File	Forced Authentication	Password Policy Discovery	Remote Desktop Protocol	Email Collection	Domain Generation Algorithms	Scheduled Transfer	Inhibit System Recovery
Trusted Relationship	Exploitation for Client Execution	Browser Extensions	Emond	Component Firmware	Hooking	Peripheral Device Discovery	Remote File Copy	Input Capture	Fallback Channels		Network Denial of Service



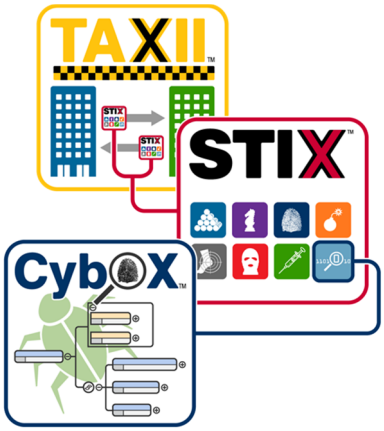
A goal of FOR508 is to educate analysts and hunters on attacker techniques and demonstrate how to find them across the enterprise. At the completion of this course, we expect you to have a very strong understanding of the most common attacker techniques and a toolkit for identifying and mitigating those techniques. The good news is the list of common attack techniques is manageable; the bad news is the set is growing and there will always be more obscure techniques discovered in the wild. The ATT&CK framework is an excellent resource for keeping up to date on attacker techniques and associated artifacts. By looking at attacks through this lens, you can identify new techniques to hunt for and judge the effectiveness of your current data sources and hunting toolkit. As an example, by mapping the "DoubleTap" malware on this slide to common techniques, you have an immediate list of items to hunt.[1] Do you have the capability to audit and stack Windows scheduled tasks across your environment? Are you aggregating command-lines and analyzing that data store for commonly used attacker commands? Are you collecting NetFlow data that can quickly identify uncommon port usage? The power of ATT&CK is it can help teams organize and prioritize their efforts while providing in-depth descriptions of techniques in use across different attacks and threat actors.

The easiest way to learn more about attacker techniques is by leveraging the wealth of open-source threat reports available. More and more vendors are mapping their threat intelligence to ATT&CK techniques. The mapping on this slide was accomplished by Katie Nickels in the excellent guide "Getting Started with Att&ck October 2019" (https://attack.mitre.org).

[1] https://www.fireeye.com/blog/threat-research/2014/11/operation_doubletap.html

Indicators of Compromise (IOC)

- A formal language allowing artifacts to be described in clear and unambiguous terms
- Facilitate information sharing
  - Describe once, find many
- Several standards
- IOCs vary wildly in efficacy
  - Home-grown often most applicable
- Tools typically drive usage



An indicator of compromise (IOC) describes attacker tools and tradecraft using a rich and precise language that can be understood by both humans and security tools. IOCs can be a particularly powerful technique to identify malware components on a compromised host. Generally, they include a combination of Boolean expressions that can be used to identify characteristics of malware. If these characteristics are found and the Boolean conditions satisfied, then you have a hit. Since finding the first hit is one of the most challenging parts of incident response, a targeted set of IOCs can greatly speed up the IR process.

There are two broad types of indicators: host-based and network-based (similar to snort signatures, plus additional data). IOCs are the equivalent to narrowing down a suspect through identifying specifics about the suspect: male, 6 ft. 2 in., ~200 lbs., shaved head, blue eyes, and driving a red or orange Nissan Xterra. Indicators of compromise are typically created by reversing malware and through application footprinting. Some mature security teams have massive IOC lists that range in the thousands of indicators collected from previous intrusions. IOCs are the difference between having to analyze each system in-depth or analyzing a few in-depth and using that data to identify similar machines on your network with the same characteristics.

Indicator Sharing
With the increasing use of threat intelligence data, several competing projects are maintained. Which indicators you use in your environment will largely be driven by your security stack. Different security tools leverage different signatures, which means you may have to provide care and feeding for multiple different indicator types. Not all IOCs are created equal, and you may find many terrible IOCs being circulated via information sharing networks. IOCs are not a "fire and forget" technology. They require active management, testing, and modification to be used effectively. The best IOCs typically are those you develop tied to your own network and via your own investigations.

STIX = Structured Threat Information eXpression
From the website: STIX™ is a collaborative community-driven effort to define and develop a standardized language to represent structured cyber threat information. The STIX Language intends to convey the full range of potential cyber threat information and strives to be fully expressive, flexible, extensible, automatable, and as human-readable as possible. All interested parties are welcome to participate in evolving STIX as part of its open, collaborative community. https://for508.com/58a14

YARA
YARA is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. With YARA, you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description (known as rule) consists of a set of strings and a Boolean expression, which determines its logic. You can find more information on YARA here: http://for508.com/-10oz

OpenIOC
OpenIOC was originally designed to enable MANDIANT's products to codify intelligence in order to rapidly search for potential security breaches. MANDIANT has standardized and open sourced the OpenIOC schema and released some basic utilities to support the standard. While you will still see indicators in OpenIOC format, its popularity has diminished within the security community. Conversion of OpenIOC to STIX: http://for508.com/ntkfy.

Two popular tools for managing indicators of compromise are CRITS (http://for508.com/st2ce) and MISP (https://www.misp-project.org/). These tools support all the above IOC formats.

Indicators of Compromise - YARA

```
rule SeaDuke_Sample {
      meta:
            description = "SeaDuke Malware"
            license = "https://creativecommons.org/licenses/by-nc/4.0/"
            author = "Florian Roth"
            reference = "http://goo.gl/MJ0c2M"
      strings:
            $s0 = "bpython27.dll" fullword ascii
            $s1 = "email.header(" fullword ascii
            $s2 = "LogonUI.exe" fullword wide
            $s3 = "Crypto.Cipher.AES(" fullword ascii
            $s4 = "mod is NULL - %s" fullword ascii
      condition:
            uint16(0) == 0x5a4d and filesize < 4000KB and all of them
}
```

1 2 3

YARA is currently the most widely used indicator of compromise format. Its popularity stems from striking the right balance of simplicity and power, making it easy for malware analysts and incident responders to identify and classify malware samples. YARA rules are written to match patterns, and the rules themselves are easily understood by both machines and human operators.

While many YARA rules are strings based, more sophisticated rules can be crafted using regular expressions, wildcards, conditions, and modules such as pe header components from the portable executable structures. In the example on this slide, the matched condition requires a 1) "MZ" portable executable signature, 2) a file size limit, and 3) matching of five specific strings found inside the file. The goal of an IOC is to create a signature that is specific enough to limit false positives at scale, while being broad enough to still match different variants of the same malware sample. This is a difficult balance and is why different IOCs have wildly different efficacy rates.

This page intentionally left blank.

Malware-ology

Attacker Tools, Techniques, and Procedures: Malware TTPs

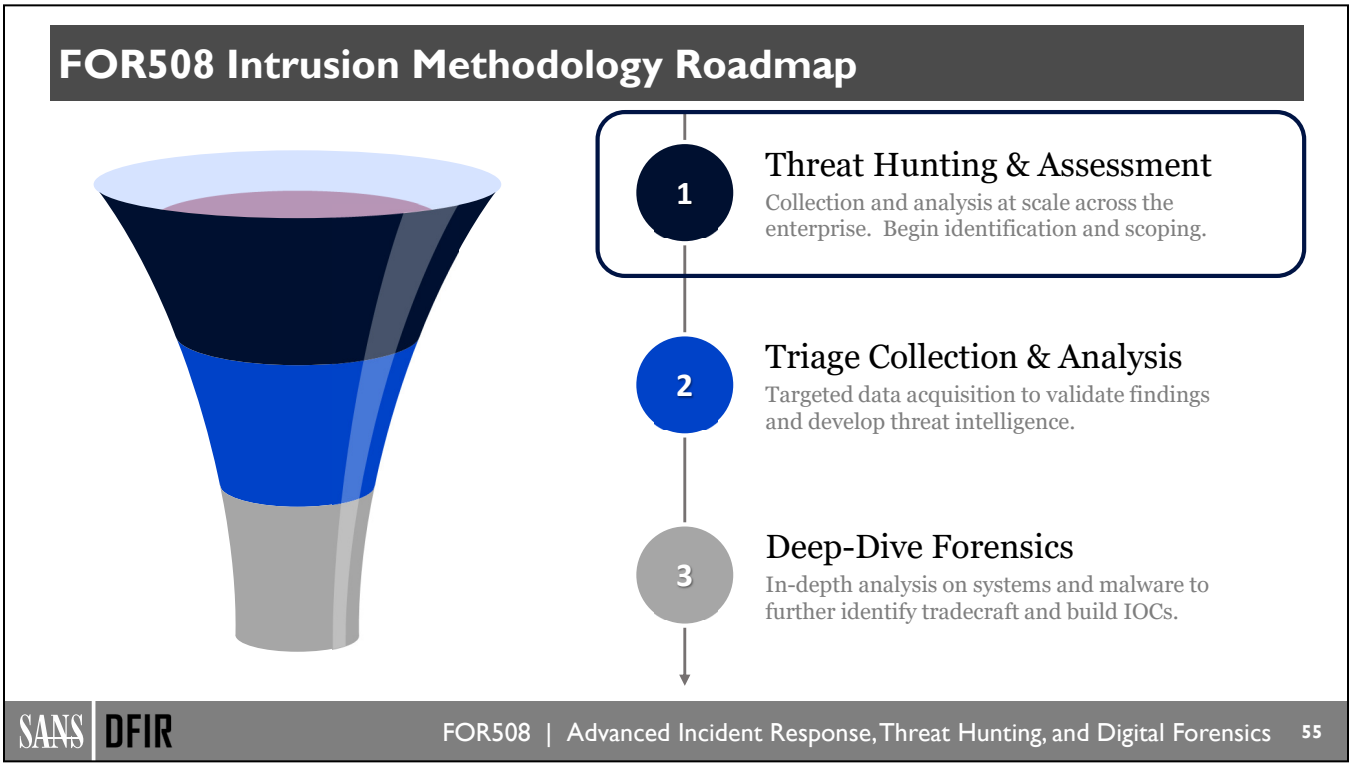
"We don't use the word 'intelligence' with software. We regard that as a naive idea. We say that it's 'complex.' Which means that we don't always understand what it's doing."
– Orson Scott Card, *Ender's Shadow*

This page intentionally left blank.

Malware Paradox

Several years ago, Jesse Kornblum stated, "Malware Can Hide, But It Must Run," and this became known as the Malware Paradox.[1] The paradox means that malware can exist but sooner or later something must activate it to run. Execution leaves telltale artifacts. As an example, methods to keep malware "persistent" across multiple reboots on a system is called a "persistence mechanism." It is a simple piece of evidence to look for and could possibly help us point, in reverse, back to the malware—more on that shortly.

[1] Exploiting the Rootkit Paradox with Windows Memory Analysis http://for508.com/-g86k

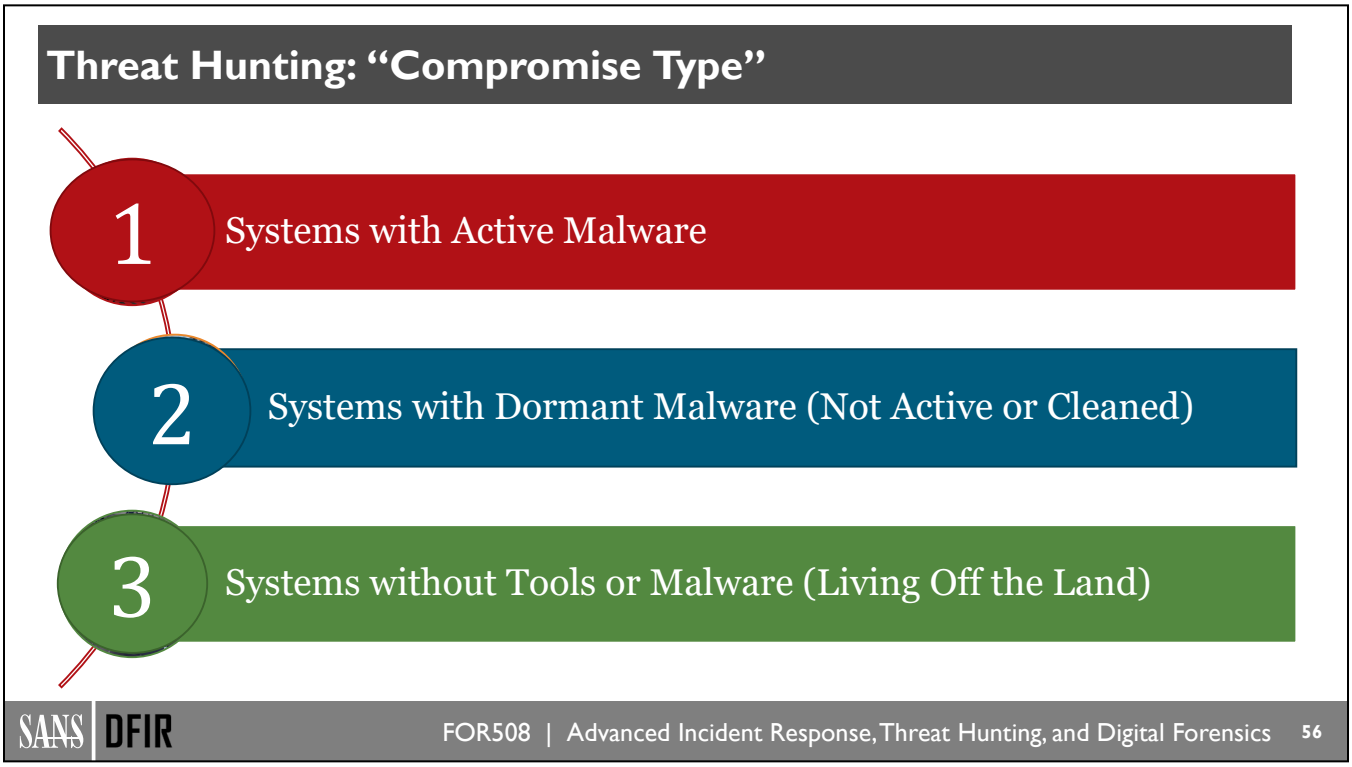


Your journey through FOR508 has been designed to follow a standard workflow for performing threat hunting, compromise assessments, and incident response activities. The roadmap we will use in this class is as follows:

Threat Hunting & Assessment
We will start our process by looking at the network using tools that can scale collection and analysis, focusing on occurrence stacking and outlier analysis. Most attendees have thousands of endpoints necessitating broad scoping techniques at the start of an investigation.

Triage Collection & Analysis
As systems of interest are identified, we will perform targeted triage collection to acquire a deeper understanding of attacker activity. Triage data can include traditional forensic artifacts like application execution data, file system information, and in-memory artifacts such as process trees.

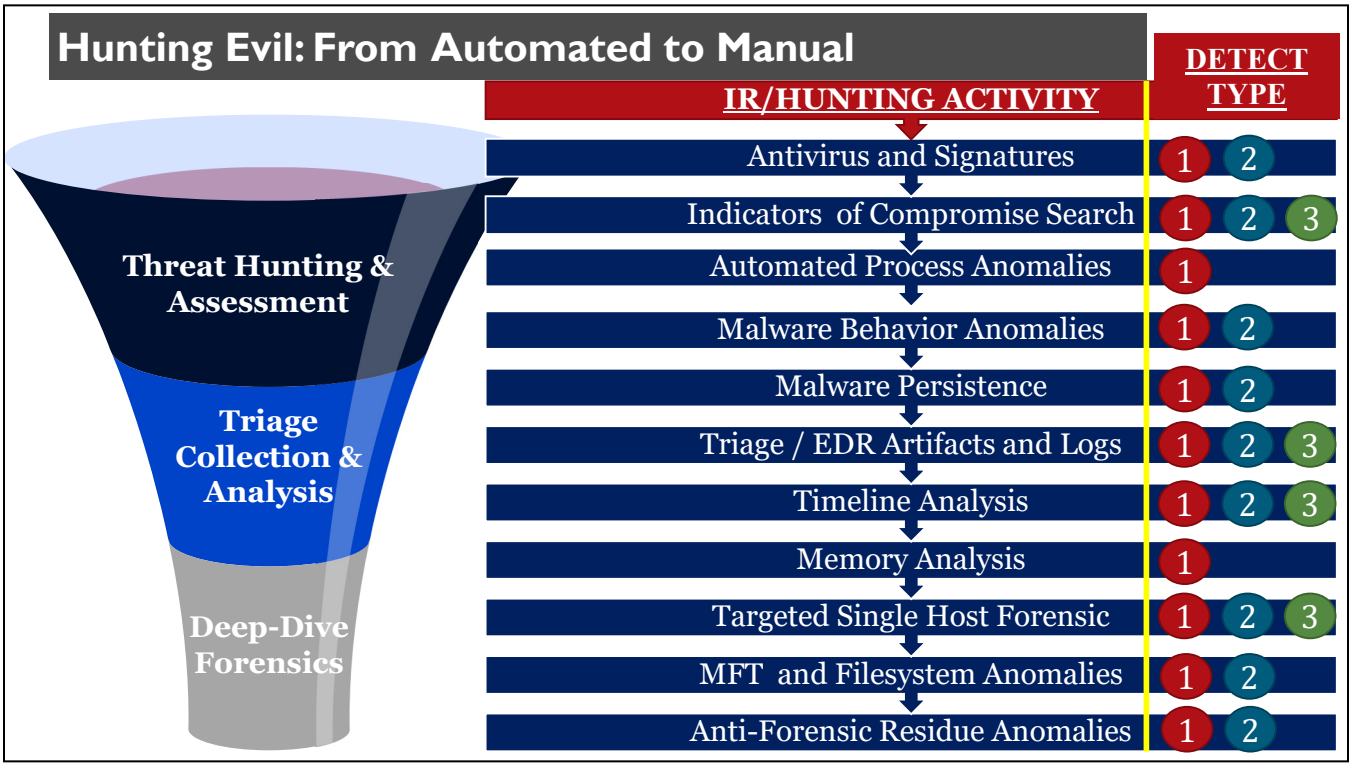
Deep-Dive Forensics
Finally, we will reserve our limited analyst time for performing deep-dive forensics on only a handful of systems having the best chance to help us understand attacker tools and tradecraft and craft better indicators to assist with scoping additional compromised systems.



Systems involved in a compromise can be largely collected into three categories, as seen on this slide. During the course of FOR508 you will see systems that fit into each category and get a good understanding of how each can be identified. We will build a kit of tools and techniques that can assist with hunting and identifying compromised systems of each type.

Three Possible Detection Types

When hunting for a compromise, active malware is often the easiest to identify. Active malware generates a wealth of artifacts, providing more opportunities for responders to identify it. Dormant malware can be much harder to detect, as we lose the recency of artifacts and ability to detect the malware in memory. Dormant malware is a broad category and could consist of a tool that was only run once, such as a credential dumper, or something executed rarely via something like Word macro or via a scheduled task. Systems that are cleaned by attackers or are detected long after the attack occurred also fall into this category. We will have to work harder to find evidence of compromise in these cases. The final category to consider are systems interacted with by the attacker, but with no tools or malware introduced. This last type is the hardest to investigation, and good attackers know this. While working through the attack cycle, there are many steps taken by attackers that do not require new tools or malware. Being able to log on to a system using valid credentials and subsequently searching for data to exfiltrate leaves a very small and difficult-to-detect footprint. We have also seen an uptick of use of "living off the land" techniques, meaning using built-in utilities to accomplish actions usually done with external tools. While detection is possible across all categories, the latter require good data collection and even better analysts!



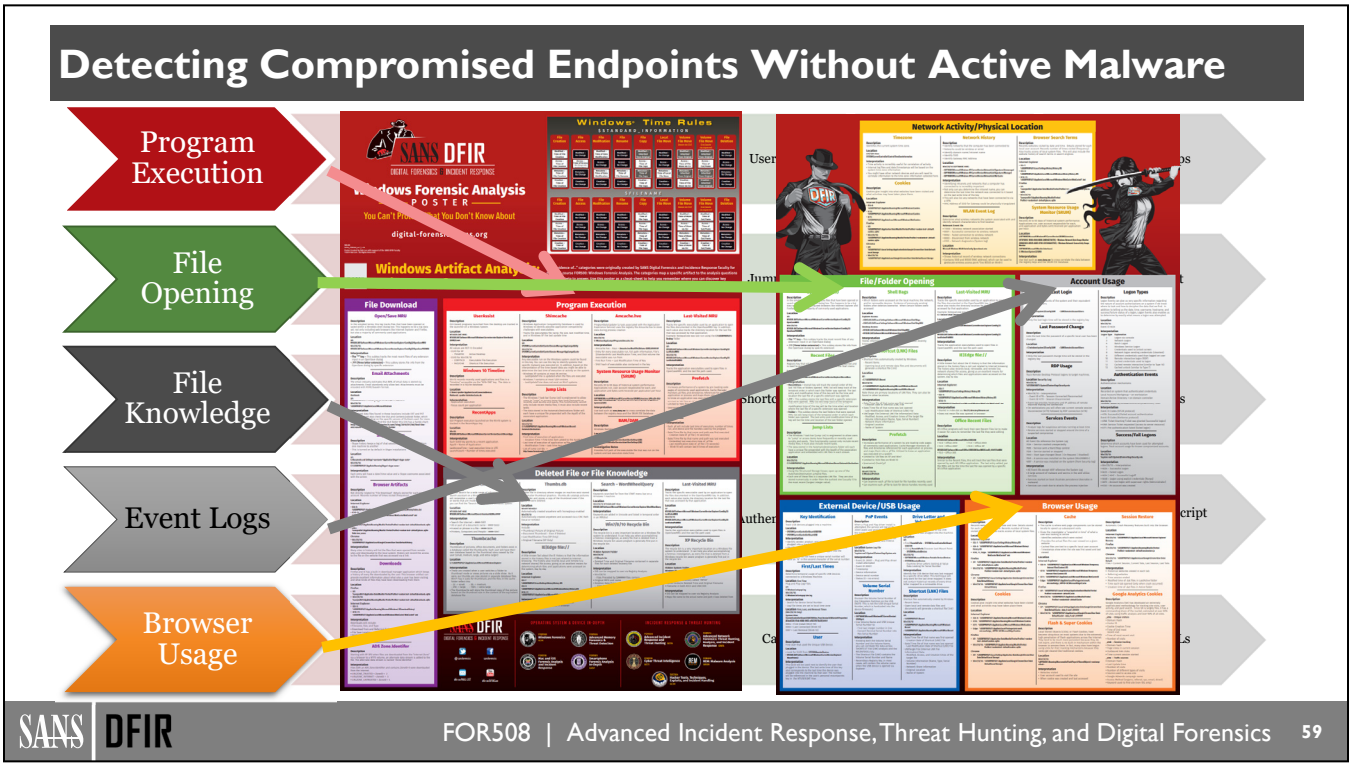
Defending a network requires balancing of limited resources across increasingly large enterprises. It is simply not feasible to perform every forensic test on every system in a network. The goal is to look for capabilities allowing scoping and analysis to be conducted at scale, while reserving the deep forensic techniques for only a small subset of systems.

As hunting or incident response begins, the focus starts with enterprise-wide collection and using automated signature detections whenever possible. These signatures could be as simple as your host-based anti-virus or more robust indicator of compromise signatures leveraging threat intelligence and tools like enterprise detection and response (EDR). As we covered earlier, some of your best threat intelligence is derived internally from deep dive and targeted triage analysis of compromised hosts. These more expensive and time-consuming operations support scoping an environment en masse. Some or all of this activity at this stage may be accomplished at your security operations center (SOC).

Triage collection and analysis begins either with alerts from the previous phase or using a hypothesis and predictions of where adversaries might appear. This stage starts to focus on standard aberrant characteristics of malware, the search for persistence, deep log file analysis, and scalable forensic artifacts showing lateral movement, data collection, and evidence of malware execution. It is hard to automate the analysis for a lot of this activity, but still possible to perform automated collection. Analysis techniques such as data stacking, frequency of least occurrence, and behavioral anomaly detection are applied. The downside is you need humans (hunters) to work through the data to identify likely candidates for deeper inspection. Triage data can be collected from a specific system or at scale across many, placing the data into large databases like Splunk or Elastic.

Finally, once an anomalous candidate system is identified, the hunter will sometimes need to take a much closer look at the system. We call this deep-dive forensics, and it allows validation of compromise and the creation of cyber threat intelligence. Indicators extracted from deep dive analysis eventually become automated signatures used to scan the enterprise at the assessment or triage phases. Deep-dive digital forensics can provide answers to hard questions like initial exploitation, how credentials were stolen, adversary intent, and data collection/exfiltration.

How does an analyst know which systems to perform deep dive analysis against? There are no hard and fast rules, but when you find a system that does not match known patterns from an adversary or is an unknown, a closer look should be accomplished. Proper analysis will fill gaps in your capabilities for cyber threat intelligence, allowing you to better track that specific adversary. It could be key not only to scoping the current intrusion, but also to finding the same adversary in future intrusions.



It is much easier to detect systems with active malware than systems with only left-over residue. Because systems cleaned or compromised with living-off-the-land techniques often require deep dive forensics, the enormity of the analysis across an enterprise is daunting. However, in practice, detection is the same as systems with active malware—find traces of attacker activity and scan the enterprise looking for those same fragments. It just turns out the fragments are harder to find on these systems. Hackers change their profile, but not often enough that their profile will be completely unique on each system. Forensic analysis is very important in this phase of the investigation to identify the patterns of attacker activity. As seen on this slide, we have an enormous set of forensic artifacts used to detect activity of interest on a system. Forensic skills for intrusion scenarios are very similar to those required for tracking other crimes.

This class assumes some familiarity with what we call, "conversational forensics." For example, if we are determining whether malware was executed on a machine, I might mention, "I looked at the prefetch folder for malware execution and also found nothing in the UserAssist key. However, I did find some evidence in the Shellbags that might be useful for our damage assessment." Conversational forensics means you are familiar with most of the terms, and although you might not have them all memorized, a quick mention of the artifact reminds you of what it tells us. If you are not at this level yet, never fear! The Windows Forensic Analysis poster from the SANS FOR500 is here to help. It is a terrific resource to reference as we go through this class and when you perform these investigations for real. Some of the techniques taught in this class become even more powerful as you develop a deeper forensic skillset. SANS FOR500 is the "other half" of this class and would be a great follow-on when you are ready to take your deep-dive forensic skills to the next level.

Adversary Hiding in Plain Sight

- Common Malware Names
  - **svchost.exe**
  - **iexplore.exe**
  - **explorer.exe**
  - **lsass.exe**
  - **win.exe**
  - **winlogon.exe**
- Common Malware Locations
  - **\Temp** folders
  - **\AppData**
  - **\$Recycle.Bin**
  - **\ProgramData**
  - **\Windows**
  - **\Windows\System32**
  - **\WinSxS**
  - **\System Volume Information**
  - **\Program Files and \Program Files (x86)**

We are going to see techniques from some of the most advanced malware toolkits on the planet this week. Rootkits, bootkits, side-loading, process hollowing, and injection are all extremely powerful means to hide malicious activity. While it is fun to "geek" out on those advanced samples, it is often just as effective for malware to simply pick a name and location and blend into the noise. In fact, the simple approach can make attacks even more survivable without the risk of creating instability, crashing processes, or getting detected with the next security tool installed. A standard Microsoft Windows system has hundreds of thousands of files and thousands of folders to house those files. And the average enterprise has thousands of these systems. Among all that noise, finding a believable name and location for your malware or backdoor to live is quite easy.

While malware names and locations are only limited by human creativity, there is commonality among malware and threat groups. As an example, svchost.exe remains one of the most popular names for malware on the planet. This is almost certainly due to it being a process found running with at least 10+ instances on modern systems, with many administrators having no idea why.[1] This slide contains some of the most common malware names and locations we have seen in the wild, but you will undoubtedly find more in your investigations. Stay curious and suspicious. Think about context. Does it ever make sense for something to execute from the $Recycle Bin? Threat reports, and resources like Virus Total, can give more ideas of places to focus (the live statistics from VirusTotal are quite interesting).[2]

[1] The typographical and homomorphic abuse of svchost.exe: http://for508.com/j2v1i
[2] VirusTotal Statistics: http://for508.com/c2-bk

Living off the Land Binaries (LOLBin)

- Legitimate binaries are increasingly used to evade security tools, allow-listing and hunting
  - `rundll32.exe "c:\kb4549947:aclui.dll",DllMain`
- Command lines provide context
  - Executions, downloads, creation of files, child processes

The LOLBAS project collects, categorizes, and provides example usage

Binary	Functions
At.exe	Execute
Atbroker.exe	Execute
Bash.exe	Execute, AWL bypass
Bitsadmin.exe	Alternate data streams, Download, Copy, Execute
Certutil.exe	Download, Alternate data streams, Encode, Decode

Built-in system binaries (executables) have been abused since the first operation systems. While their use is nothing new, we have witnessed a large growth in the volume of illegitimate usage in the Windows world. This growth is an extension of the "living off the land" approach to attacks and is a direct result of host-based security tools getting better at detecting and preventing malicious code from executing. Imagine an application control product that is highly reliant on digital signatures. Instead of using a home-grown tool (unsigned) to copy a locked credential file from a sensitive folder, why not use one of the many built-in Windows tools (digitally signed by Microsoft) to accomplish the task? As research into the use of native Windows tools has increased, the number of creative and surprising use cases of standard Windows tools has exploded. The LOLBAS (Living Off the Land Binaries and Scripts) project was started by Oddvar Moe and does an excellent job of collecting and categorizing relevant attacker use cases for legitimate Windows binaries.[1] This is a great place to familiarize yourself with the various binaries and the myriad of ways they can be abused. Better yet, try some of the techniques on your own system so you can better understand how they could be useful to an attacker!

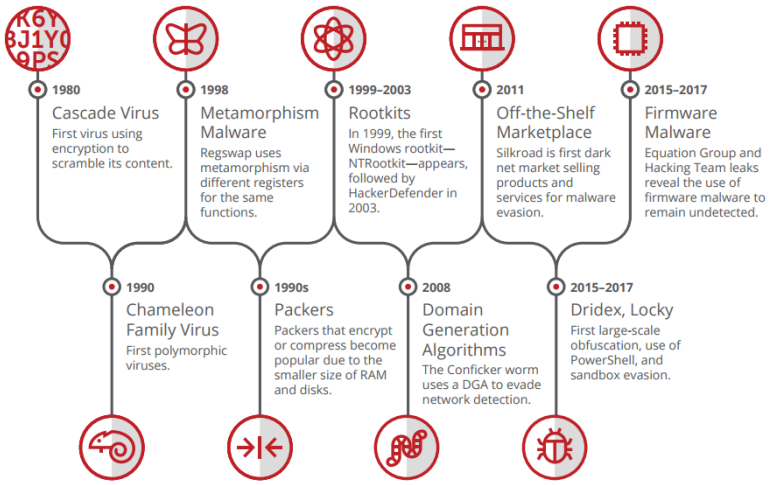
Many of the examples on this slide are extremely popular in the wild. Bitsadmin.exe and Certutil.exe are used extensively by attackers for stealthy file downloading. We also regularly see Certutil.exe abused to decode obfuscated payloads, attempting to avoid host-based security. Rundll32.exe is a legitimate means to run code present in a DLL. It requires the name of a DLL, the function name to execute within the DLL, and any additional arguments. Since this binary is legitimately used in Windows to run code, you can see why it might be difficult to weed out legitimate versus illegitimate uses. From a defender's perspective we need more information than just "rundll32.exe" executed. We need the context that comes along with the full command line and arguments, along with the DLL that was executed (if possible). In the example on this slide, the DLL loaded is in an alternate data stream (notice the colon in the filename), which is highly unusual and a strong indicator for suspicion. Knowing if there were any files downloaded or created by the tool, and what child processes were spawned in memory could also be very helpful. The fact that tools like memory forensics and EDR (endpoint detection and response) give this context is a big reason they have become so useful to the modern defender.

[1] LOLBAS Project: https://for508.com/pahr5

Malware Common Defense Evasion Techniques

- Service Hijacking/Replacement
- Process Injection
- Filename/Service Hijacking
- Alternate Data Streams
- WebShells/Beacons
- Firmware
- DLL Injection
- A/V Bypass
- Frequent Compilation
- Binary Padding
- Packing/Armoring
- Dormant Malware
- Signing Code with Valid Cert
- Anti-Forensics/Timestomping
- Rootkits
- "Fileless" Malware

Major Milestones in the Evolution of Evasion Techniques



Source: McAfee, 2017.

Hiding in plain sight is one successful technique employed by malicious software, but certainly not the only one. There are myriad other ways malware authors have devised to stay hidden and accomplish the mission. This slide contains many of the common techniques, and we will see and talk about most of these throughout the course. An important concept to understand is malware authors must make decisions, and there is never a perfect choice. Process injection may be very stealthy to many of the API-based security and analysis tools but be trivial to identify with memory introspection. Packing or armoring an executable can easily evade most anti-virus products but creates a highly anomalous file easily found in other ways, such as identifying the amount of compression, encryption, or unusual sections within the executable. Signed code might help evade security and allowlisting solutions but becomes ineffective once that code signing certificate is identified as evil and revoked. Just like in physics, every action has an equal and opposite reaction. We use the signatures and artifacts created by these various evasion techniques to find evil. Some of the artifacts can be quite subtle and difficult to parse from normal activity. This is exactly why threat hunting is so important. Hunters can perform analysis and create interesting patterns to find new techniques. You will learn many of these techniques this week.

Trusted code signing is intended to increase the security and trustworthiness of programs downloaded from the Internet. In the past, when most programs were bought at physical stores, programs were inherently trusted because they came in original packages that were shrink wrapped. Code signing is intended to produce the same trust in programs purchased and downloaded across the Internet. It is fairly trivial to "self-sign" code. However, trusted code was intended to be a different story. With signed and trusted code, one wouldn't have to worry about it being malicious—or at least that was the intent. In practice, signed malware is a real threat and has been seen multiple times in highly publicized incidents.

How is trusted code signing supposed to work? A certification authority, such as Verisign or Thawte, verifies and confirms identity information, issuing a valid digital certificate to the organization or individual. The intent of the certification is to identify the original code author. Once issued, the developer can use his or her private key to digitally "sign" the code, providing attribution. Authentication of organizations and/or individuals applying for a code signing certificate varies widely. For some companies, you need to provide a copy of your passport or even a phone bill and the certification authority will call the number for verification.[1] In other instances, authorities may ensure you are not on a restricted organization list from the government, have a business license, exist at a physical address, have rights to the domain name, and are verified by a third-party phone number. To obtain a commercial software certification, most organizations must also apply for a Dun & Bradstreet Rating. According to MSDN, "Applicants must achieve a level of financial standing as indicated by a D-U-N-S number (which indicates a company's financial stability) and any additional information provided by this service. This rating identifies the applicant as a corporation that is still in business. Corporations that do not have a D-U-N-S number at the time of application (usually because of recent incorporation) can apply for one and expect a response in less than two weeks."[2]

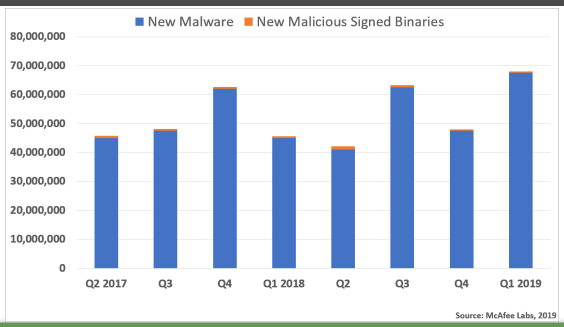
The Dun & Bradstreet rating is intended to draw a very hard line to cross between commercial and private (individual) developers. Although not impossible to create a front company to meet this criterion, the process is lengthy and tedious. While achievable for criminal syndicates and nation-state adversaries, it may be less likely for less resourced entities. An alternate route, chosen by some malware developers, is simply to steal commercial-level code-signing private keys from someone else. This can be as easy as finding a private key inadvertently added to a public Git repository. We have seen multiple instances of code-signing certificate theft

in the wild. Flame and Stuxnet are two high profile examples, but we have also seen code signing certificate private keys stolen from Adobe and used to sign malicious software.[3] An attack on the company behind the Opera browser allowed the compromise of their code signing private key, which was subsequently used to sign malware.[4]

Code signing provides the ability to trust that code has not been tampered with after signing it, in addition to aiding in identifying the origination of the code. Major operating systems will require code to be signed by a trusted developer in order to allow them to execute without user interaction. In some of the latest Microsoft server products drivers must be signed in order to load. Signed malware has an easier time spreading and hiding on networks, but it does come with significant cost. If the malware is ever discovered, the code signing certificate could be revoked and added to a Certificate Revocation List (CRL). The CRL list should be regularly checked and any system with that software installed would no longer function. However, in practice, this process has produced faulty results and has required major patches to Microsoft operating systems to update the CRL list. Windows is working to improve this process and enforce even more signing requirements in future updates. While code signing is still relatively rare for malware, it may very well be a requirement in the future. Of course, there are always work arounds. We will likely see more advanced techniques like DLL Side-Loading developed and employed to get around these restrictions.

[1] Enrollment Guide for Thawte Code Signing certificates (PDF): http://for508.com/1srk6
[2] Introduction to Code Signing: http://for508.com/oca3f
[3] Inappropriate Use of Adobe Code Signing Certificate: http://for508.com/a4u0k
[4] Stolen Opera Code-Signing Certificate Used to Sign Malware: http://for508.com/b5eon

Will Malware Be Signed?



96%

of malware is unsigned

Windows Defender ATP, 2019

- McAfee reports between 1-3% signed malware. Microsoft Defender reports 4% signed
- Know your adversary—Nation-state threat actors have a higher percentage of signed code
- Don't ignore signed code but consider focusing first on unsigned programs
- Reduce datasets via trust of well-known companies: Microsoft, Apple, and Google
- Signed programs in "system locations" not from a well-known entity are suspicious

Will Malware Be Signed? It depends. There are benefits and drawbacks to signing code. According to the McAfee Labs 2019 Threat Report, much less than 3% of all malware is signed. In some quarters it was below 1%. In fact, the bar chart on this slide barely registers the "New Malicious Signed Binaries" because the numbers were so low versus total malware identified each quarter. These percentages are consistent with the past several years. Malware signing is predicted to increase over time, but the fact is that the total number of signed malware samples compared to unsigned malware is very small. We can use this to our advantage as we hunt for malware across our enterprise.

Benefits to Signing Malware

Signed malware is trusted by the operating system and can stay hidden for a longer period without arousing suspicion. Espionage malware is a classic example, bringing to mind the infamous Flame malware, intended to stay hidden for as long as possible. Signing is advantageous if the developer is not planning on using that malware again and is willing to risk it being revoked if discovered.

Drawbacks to Signing Malware

Rapid development and release of malware will be inhibited by signing requirements. Malware authors often need to rapidly develop alternatives to their code to avoid antivirus and host-based intrusion detection systems. A malware author would need a plethora of code signing certs to avoid burning an entire family of malware active across an enterprise if discovered. If a code signing cert is used and later revoked, all the current locations of the malware will be flagged, becoming a major liability and easy means of detection. For malware that is rapidly released, adversaries might have only a limited number of code signing certificates available and malware developers might be supporting engagements across multiple targets. If certificates are re-used and one of the samples is discovered and revoked, it could burn all the malware currently installed across those targets. Even nation-states do not have unlimited resources and we have regularly seen these mistakes made in the wild. This is one of the reasons it can be so lucrative for defenders to focus on digital signatures.

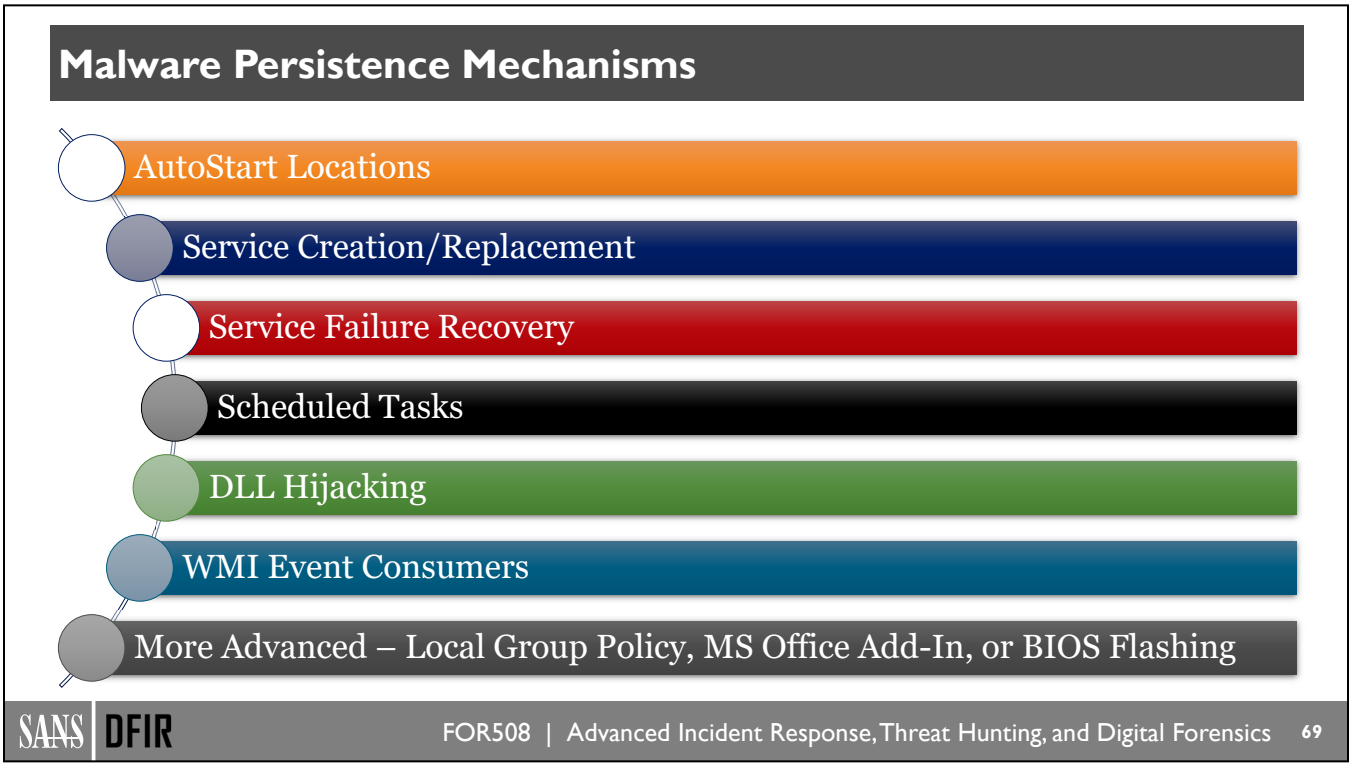
The Bottom Line:
There is still a chance that malware is signed. As a result, both signed and unsigned code should be examined. However, it makes sense to first examine all suspicious unsigned code and any signed code from unrecognized developers. You can eliminate the vast majority of signed code by initially trusting signed code from well-known companies such as Microsoft, Apple, and Google. We can also layer our detections by looking for unsigned code in unusual locations. As an example, the Windows\System32 folder primarily contains files signed by Microsoft. If you see something in that folder on a modern 64-bit operating system, you may have just found malware!

This page intentionally left blank.

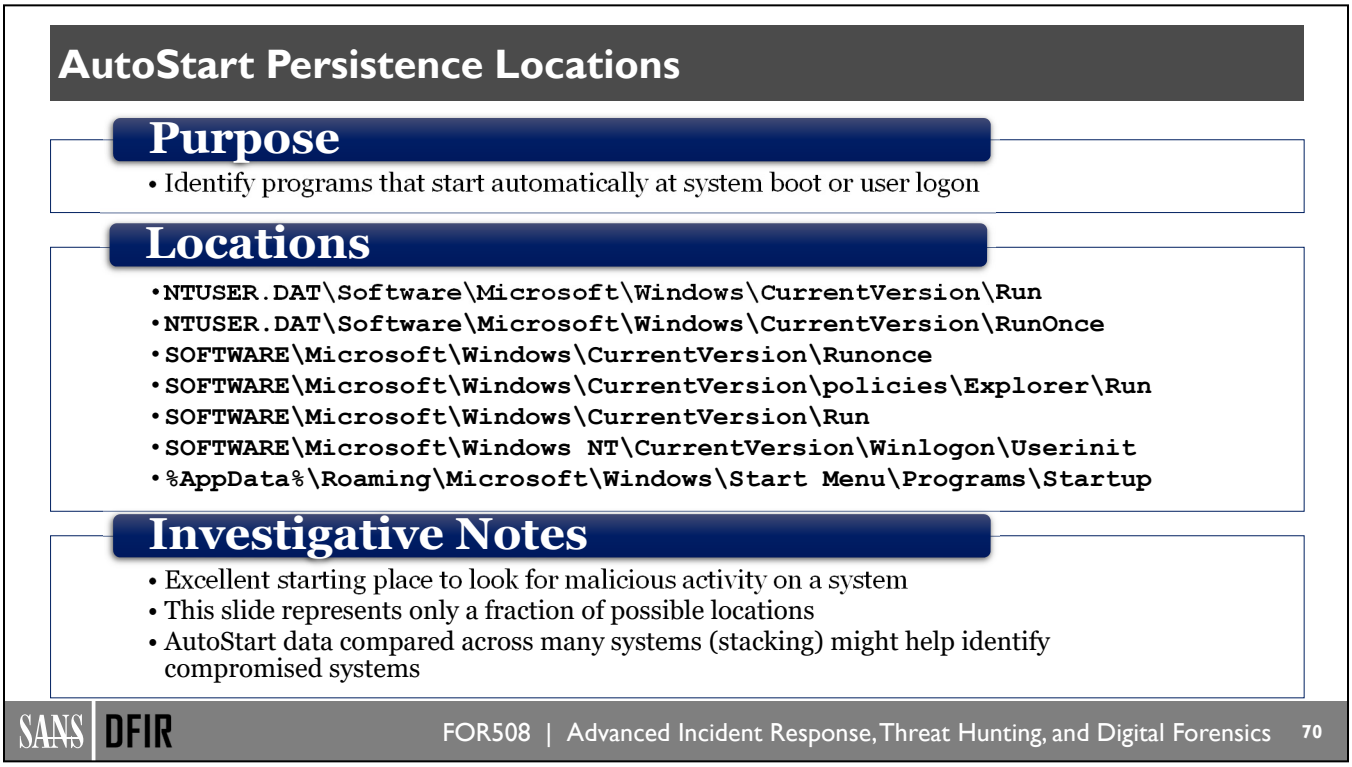
Malware Persistence

Malware wants to hide, but must survive a reboot

This page intentionally left blank.



A majority of intrusions include at least some form of persistence. Most intrusions require long-term access to achieve the attacker's goals, making persistence something implemented early in the attack cycle. This is exactly the reason it is a lucrative place for defenders to look for evil. We know it must be accomplished by the attacker, and there is a relatively small set of common persistence mechanisms (actually there are well over 50 in the Windows eco-system, but even this is a manageable number and only a small percentage are regularly used). Another big benefit to finding evil persistence is it may allow us to identify attackers early in the kill chain. Indicators early in the kill chain are sought after because the sooner you identify and remediate an intrusion, the less cleanup required, and the less likely attackers are to have achieved their goals. We have chosen to expand upon the most commonly seen persistence mechanisms for this section of the course. By some estimates, including Mandiant's M-Trends report, the first two (AutoStart locations and services) comprise over 80% of all persistence seen in the wild.



AutoStart Persistence Locations

There are a daunting number of "autorun" locations available in Windows. In Microsoft-speak, these are also known as AutoStart Extension Points (ASEPs), and they are one of the key reasons why Windows is so hard to secure. A quick look at any number of blogs shows well over 50 ASEP locations that a malicious file can place a reference to itself to ensure it survives a reboot. Luckily, many of the most common ASEP locations are in the Registry, at least giving us a single place to look (even if there are over 500,000 Registry keys on a standard system). A sampling of some common ASEPS is seen on this slide.

By far, the most popular ASEPs on the planet are the "run" Registry keys:

```
NTUSER.DAT\Software\Microsoft\Windows\CurrentVersion\Run
NTUSER.DAT\Software\Microsoft\Windows\CurrentVersion\Runonce
Software\Microsoft\Windows\CurrentVersion\Runonce
Software\Microsoft\Windows\CurrentVersion\policies\Explorer\Run
Software\Microsoft\Windows\CurrentVersion\Run
```

Items listed in these keys are executed when a user logs on—not at boot like other ASEPs. There is no specific order to the startup, and multiple "run" keys exist in both the NTUSER.DAT and the SOFTWARE hives.

Less common, but equally lethal, is the Userinit key.

```
SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit
```

Typically, we would expect this value to contain only a reference to userinit.exe. By default, Winlogon executes Userinit.exe and launches Explorer.exe. However, the key can be modified to include a reference like C:\Windows\system32\userinit.exe,C:\Temp\winsvchost.exe and the malicious binary will also be executed at boot.

A final location to highlight is in the file system, not the Registry. This can actually be advantageous to an attacker because creating persistence here does not require administrator rights. Even advanced attackers often use this location as an early-stage persistence mechanism via phishing attacks.

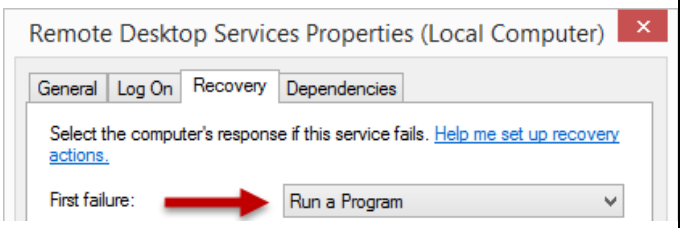
%AppData%\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

Any shortcuts created in this folder will execute the representative binary upon user logon—easy and effective!

Although these locations are very common for ASEPs, there are many, many more. Free forensic tools like Registry Explorer and RegRipper have plugins to retrieve many common ASEPs from Registry hives. We will shortly introduce tools like Autoruns and Kansa that provide the ability to collect this information at scale. Further, by collecting this data across many systems, frequency analysis can be conducted to help identify outliers that can lead us to compromised systems.

Windows Services

- New Service Creation
  - `Start` value set to 0x02 will start automatically
  - IPRIP: RIP Listener Service (APT1)
- Service Replacement
  - Modify and autostart a service to load new binary
  - GlassRAT (China) used the disabled "RasAuto" Service
- Service Failure Recovery



Windows Services
Windows services are designed to run applications in the background without user interaction. Many services are required at system boot, including the DHCP Client, Windows Event Log, Server, and Workstation services. These services provide critical functionality for the OS and must be started immediately without requiring user input. Services can be implemented as standalone executables or loaded as DLLs. To conserve resources, many service DLLs are grouped together and run under a smaller set of svchost.exe instances. svchost.exe is a Windows-generic service host process, and it is typical to see several running instances of svchost.exe (five or more is common). Service configurations, as well as device driver configurations, are stored in the Registry under HKLM\SYSTEM\CurrentControlSet\Services. The keys here provide the parameters for each service, including the service name, display name, path to the service's executable image file, the start value, required privileges, dependencies, and more. Each service has a start type value configured to start at boot, by manual intervention, or on trigger events such as obtaining an IP address or hardware device connections. Start values of 0x02 (Automatic) and 0x00 (Boot start of a device driver) can provide persistence for malicious code. Windows services provide great flexibility to developers, and similarly malware authors, for automatically running code on a Windows host.

Because services can be configured to reliably start at boot (often before the loading of antivirus), they are a very popular persistence vector. It also helps that the average Windows system can easily have over one hundred services registered, making it very easy to hide in plain sight. With administrator rights, it is trivial to either modify the Services Registry key or use the built-in "sc" command to create a service that auto-loads a malicious DLL or executable. One of the classic APT examples of this technique is to add (or replace) the rarely used RIP Listener Service (IPRIP) and use it to load a malicious executable. To give an example of how prevalent this technique is, out of the 44 malware families enumerated in the Mandiant APT1 report, 14 used Services for persistence.[1]

Service replacement is similar to service creation, but instead finds a current service that is disabled or unneeded and replaces the existing binary with a malicious one. If the service is not already set to autostart, it is trivial to

modify the start type value for the service. This can be stealthier because it is normal for that service to be on the system but requires finding an unimportant service to replace. This technique is not as common as simple service creation due to the increased complexity. The GlassRAT backdoor employed by a China-based APT group has been known to use this technique, abusing the frequently disabled “RasAuto” service.[2]

Even less common, but potentially more stealthy, is using the service recovery mode option to load a malicious binary when a specific service crashes. The example on this slide shows the Remote Desktop Services recovery options set to run a program upon failure (typically, it would default to restarting the service). As Mark Baggett points out in his post on the topic, this service might be particularly interesting because there are known vulnerabilities for reliably crashing the RDP service.[3]

The Autoruns tools from Sysinternals provides an easy means to collect and analyze services on a system. Alternatively, on live systems use the built-in “sc” command to query installed services, using parameters such as “queryex,” “qc,” “qprivs,” and “qtriggerinfo” to get detailed information on service configurations. For offline analysis, investigate service configurations within the Registry. Few tools collect service failure recovery information, but the Kansa PowerShell framework has a script named **Get-SvcFail.ps1** to collect it within its default ASEP modules. Investigating unusual service crashes in the event logs might also provide clues.

[1] Mandiant APT1 samples categorized by malware families: http://for508.com/28vic
[2] Peering into GlassRAT (PDF): https://for508.com/cw1u-
[3] Wipe the drive! Stealthy Malware Persistence: http://for508.com/ojzea

Scheduled Tasks

- **`at.exe`**
  - Deprecated, but still present in WinXP and Win7+
  - Use recorded in **`at*.job`** files and **`Schdlgu.txt`** (XP)
- **`schtasks.exe`**
  - Activity logged in Task Scheduler and Security logs
- Remote capabilities are commonly used for lateral movement and credential dumping
- The SUNSPOT implant that inserted the backdoor into SolarWinds builds used a scheduled task for persistence

Scheduled Tasks
Scheduled tasks provide an extremely granular means to create persistence in Windows. The at.exe command has long been a core part of the hacker lexicon, most notably because in WinXP, it provided a very reliable privilege escalation attack ("at" jobs originally ran as SYSTEM regardless of the user's privilege level). Even with that vulnerability patched, they are still commonly used by adversaries in Windows 7 environments, likely due to familiarity or laziness (schtasks.exe requires more typing). When an "at" job is created, corresponding ".job" files are created in the \Windows\Tasks and \Windows\System32\Tasks folders (the latter directory was added in Windows 7 and records duplicate task information in XML format). These files are named sequentially starting with "at1.job" and record details about what was scheduled. In XP, task information is also stored in the C:\Windows\Schedlgu.txt log. A simple command might look like: **`at.exe 22:01:00 c:\temp\winsvchost.exe`**

The schtasks.exe tool is an upgraded version of at.exe and has an immense number of features allowing tasks to be set and finely controlled. Tasks can even be set for specific Windows events, such as when a specific user logs on, allowing much more creativity for persistence over just using specific times. New logging for scheduled tasks appeared in Windows Vista, including a dedicated event log named "Task Scheduler Operational." A simple command might look like:
`schtasks.exe /create /sc daily /tn winsvchost /tr c:\temp\winsvchost.exe /st 22:01:00`

Interestingly, both commands have the ability to schedule tasks on remote systems. As you might imagine, this opens interesting possibilities for attackers. Remote scheduled tasks are routinely used to spread malware (including backdoors), execute batch scripts, and perform routine actions like credential dumping across many systems. Most forensic artifacts for these remote tasks will be present on the systems they were executed on, not the originating system.

It is shocking how often such a simple persistence mechanism is used even in advanced attacks. During the SolarWinds supply-chain investigation, investigators discovered the SUNSPOT malware responsible for inserting the backdoor into the software builds was executed by a scheduled task set to start at system boot time.

The Autoruns tool from Sysinternals will collect currently scheduled jobs from the task scheduler service and later in class we will explore the comprehensive logging of tasks provided by Windows.

DLL Hijacking Attacks

- DLL Search Order Hijacking
  - Place malicious file ahead of DLL in search order
  - Classic example is **`Explorer.exe`** loading bad **`ntshrui.dll`**
- Phantom DLL Hijacking
  - Find DLLs that applications attempt to load, but either don't exist or can be replaced
  - **`fxsst.dll`** (Fax Service)
- DLL Side-Loading
  - WinSxS mechanism provides a *new* version of a legit DLL
  - PlugX, NetTraveler, Sakula, Poison Ivy (RATs)
- Relative Path DLL Hijacking
  - Copy susceptible .exe and corresponding bad .dll to location of choice

DLL Hijacking Attacks

DLL persistence hijacks attack legitimate and legacy features of the Windows operating system. Search order hijacking is an excellent example. It turns out that when an executable runs in Windows, it is not required to hardcode the location of any required dynamically loaded libraries (DLLs). Instead, a specific search order is often used to find the required DLL, starting with the local directory the .EXE is run from and eventually ending up in a folder like C:\Windows\System32 where most standard DLLs exist. The only real exception is for DLLs present in the KnownDLLs Registry key that does effectively hardcode a small number of specific system DLL locations. If adversaries can find an executable that is not located in the System32 folder and loads a DLL not present in the KnownDLLs Registry key, they can place a malicious DLL in the same folder as the target executable and trump the search order, ensuring their bad code is loaded whenever that application starts. Amazingly, there are lots of these susceptible locations going back all the way to Windows 2000. A classic example was documented in the wild by Mandiant where Explorer.exe (the Windows GUI desktop) loaded a vulnerable DLL (not protected by KnownDLLs) named "ntshrui.dll."[1] Because Explorer.exe is located in the \Windows folder and the legitimate ntshrui.dll is located in the \Windows\System32 folder, all the attackers had to do is find a way to drop their malicious dll (named "ntshrui.dll") into the \Windows folder, and it would be executed every time the desktop was started. Winning!

Because this is directly related to backward compatibility, there is no likely fix on the horizon. File system forensic analysis looking for newly created DLLs in unusual locations is usually the best way to discover it. The following is a search order common to modern systems with the option SafeDllSearchMode enabled (enabled by default)[2]:

1. DLLs already loaded in memory
2. Side-by-Side Components
3. KnownDLLs list
4. Directory from which application is loaded
5. C:\Windows\System32
6. C:\Windows\system
7. C:\Windows
8. Current Directory
9. System %PATH%

Phantom DLL hijacking is a similar attack but uses the fact that some very old DLLs are still attempted to be loaded by applications even when they are completely unnecessary. In fact, some applications try to load very old DLLs that no longer even exist on modern Windows operating systems! If attackers can find such a DLL (and many are already documented), all they have to do is provide a malicious file with the same name of that long-forgotten DLL in the search path, and code will be executed.[3] Similarly, even if the DLL exists, it might be able to be easily replaced with a trojanized version. A great example is the replacement of the fxsst.dll (Fax Service) DLL in the System32 folder documented by Mandiant and still being used by attackers in the wild.[4]

DLL side-loading has a similar sounding name but is actually quite different than the previous two attacks. This attack uses the Windows side-by-side (SxS) DLL loading mechanism to introduce an "updated" version of a DLL. SxS functionality is a legitimate feature of Windows and is used by many applications to prevent problems that can arise due to updated and duplicate versions of DLLs. SxS gives the ability to load updated DLLs, but has few validity checks for these new DLLs and thus the loading mechanism can be abused due to missing DLLs, use of relative paths, and other shortcuts not taken into account by the application developer. This attack is often used to circumvent AV protections and provides an opportunity for a known good, even digitally signed, executable to be used as the persistence mechanism. The most notable example is the very popular PlugX RAT, which drops a legitimate executable (with a hash present in the NSRL known good hash database) and then uses SxS during runtime to load and construct a malicious DLL in memory. Amanda Stewart at FireEye wrote an excellent whitepaper describing the process.[5] PlugX is not alone—a wide variety of remote access tools, such as NetTraveler, use similar techniques.[6] Similar to other DLL persistence attacks, the best way to discover this behavior is to identify new executables and helper files added to the system during the attack (PlugX often creates three new files on the system).

Finally, we have the simplest option, relative path DLL hijacking. This attack could also be called "bring your own executable". Write permissions on sensitive folders like Windows and Windows\System32 are required to accomplish some of the more interesting search order hijacking attempts. Those folders may also be more closely watched by host-based security mechanisms. So, an alternative is to just copy a susceptible executable from one of the protected folders to any writable location. Then add the bad .DLL to be loaded in the same folder, and voila, you now have an incredibly easy way to get the bad code loaded. While the list of susceptible system binaries numbers in the hundreds and this attack is very easy to accomplish, the downside is you will now have a system executable (and likely persistence mechanism) running from the wrong location (wherever it was copied to). This attack is one of the most commonly used today (groups like APT32 are famous for using it, as are some variants of Poison Ivy and PlugX malware) and should be one of the easiest for analysts to identify with a combination of file system timelining and memory process analysis.

[1] Malware Persistence without the Windows Registry: http://for508.com/v2su7
[2] Dynamic-Link Library Search Order - Microsoft Docs: https://for508.com/9j5np
[3] Phantom DLL Hijacking: http://for508.com/m4frx
[4] What the fxsst?: http://for508.com/sgeyk
[5] DLL Side-Loading: A Thorn in the Side of the Anti-Virus Industry (PDF): http://for508.com/z8ni0
[6] NetTraveler APT Targets Russian, European Interests: http://for508.com/78nj2

Hunting Notes: DLL Hijacking

DLL hijacking is excellent for persistence but is also commonly used to evade host-based security and to achieve privilege escalation

- File system analysis: Look for new or unsigned .exe/.dll files in unusual places

Timestamp	macb	Meta	File Name	File Size
2021-02-18 03:42:31	.a.b	39301-128-4	c:/ProgramData/mcoemcpy.exe	77824
2021-02-18 03:42:31	.a.b	39303-128-1	c:/ProgramData/McUtil.dll	131072

- Memory Analysis: Find system processes or DLLs loaded from wrong location

```
mcoemcpy.exe pid:   2463
Command line : \??\C:\ProgramData\mcoemcpy.exe 0x4

Base                  Size  LoadCount  LoadTime                      Path
------------------ ---------- ----------  ------------------------------ ----
0x00007ff6f5ba0000    0x11000     0xffff  2021-02-30 22:15:18 UTC+0000   C:\ProgramData\mcoemcpy.exe
0x00007ff83df00000   0x1e0000     0xffff  2021-02-30 22:15:18 UTC+0000   C:\WINDOWS\SYSTEM32\ntdll.dll
0x00007ff83b3d0000    0xae000     0xffff  2021-02-30 22:15:18 UTC+0000   C:\WINDOWS\System32\KERNEL32.DLL
0x00007ff83a250000    0x1f000        0x6  2021-02-30 22:15:18 UTC+0000   C:\ProgramData\McUtil.dll
```

Legit McAfee

Evil DLL

- Hijacking is often paired with other tradecraft like code injection and command and control network beaconing that can also lead to detection

DLL hijacking can be a very stealthy attacker technique but is often easily defeated with thorough forensic analysis. Nearly all DLL hijacks require placing a new DLL and/or executable onto the file system. This presents a useful detection point since newly created DLLs and executables are rare on most systems. Paying attention to newly created files around a time of interest is a classic forensic timelining technique that we will see later in the class. Memory forensics is another technique well suited to finding code running from unusual locations since all legitimately loaded code must come from disk. By looking for processes that should not be running, or those loading DLLs from strange locations, an analyst could easily find evidence of DLL hijacking in memory. Also note that hijacked DLLs tend to be the more obscure ones because the most common DLLs are already loaded in memory, preempting the loading of any evil versions with duplicate names (Windows first checks if a DLL is already loaded in memory before looking on disk). Added to this is that code loaded through hijacking almost always performs other actions likely to draw the eye of an investigator. These could include reaching out over the network, creating named pipes, or injecting code into other processes. We will have many ways throughout the course to detect anomalous actions and any of them could lead back to a DLL in a strange location, and ultimately help you discover a DLL hijack attack.

The examples on this slide show evidence of a relative path DLL hijack attack commonly tied to the Ocean Lotus/APT32 threat actor. The executable mcoemcpy.exe is legitimate, digitally signed software from McAfee. It has been copied to an unusual location (ProgramData folder) along with a malicious DLL, McUtil.dll. Upon execution, mcoemcpy.exe also loads McUtil.dll. A keen analyst might wonder why these newly created files were present in the root of the ProgramData folder, a location not typically known to hold executable files. If some of the tool output on this slide is unfamiliar to you, don't worry, we will be covering them in depth in later sections.

WMI Event Consumer Backdoors

WMI allows triggers (filters) to be set that when satisfied will run scripts or executables

1. **`Event Filter`** → Trigger condition
2. **`Event Consumer`** → Script or executable to run
3. **`Binding`** → Tie together Filter + Consumer

- Filters can be based on time (every 20 sec), service start, user auth, file creation, etc.
- PowerShell or mofcomp.exe can be used for setup
- The PowerShell cmdlet "**`Get-WmiObject`**" can identify and help remove suspicious entries

WMI Event Consumer Backdoors

WMI is often overlooked by security professionals, but it contains very powerful capabilities that have not gone unnoticed in the hacker community. One of the more recent persistence methods identified in the wild has been the use of WMI Event Consumers. WMI provides the ability to monitor for specific events and when triggered, alert event consumers that can then do things like run scripts and execute code.[1] Administrative privileges are necessary, but once achieved, attackers can use WMI to create a backdoor that is difficult to detect without the proper tools. WMI consumers run with SYSTEM privileges.

The technique requires three discrete steps:

1) An event filter must be created describing a specific trigger to detect (for example, trigger every twenty seconds).
2) An event consumer is added to the system with a script and/or executable to run (run a PowerShell script to beacon to a command and control server).
3) Finally, the event and consumer are tied together via a binding, and the persistence mechanism is loaded into the WMI repository.

The three steps are often written inside a managed object format (MOF) file that is used to register new classes into the WMI repository. To be even more stealthy, PowerShell **`Set-WmiInstance`** or **`CreateInstance`** can also be used. Event filters can be set up to trigger immediately upon being registered or via virtually any other windows event such as a specific time, the existence of a file or folder, service starting or stopping, a specific user being authenticated, etc.

This type of attack is not theoretical. Stuxnet was perhaps the first sample in the wild to use the attack.[2] It used a zero-day vulnerability in the print spooler (MS10-061) to allow the transfer of two files to remote systems—an .EXE and a .MOF file. The .MOF file was auto-compiled by the system, creating a WMI event filter and consumer to immediately execute the .exe file. Wow! Now modern malware samples like ransomware and crypto-worms have picked up where Stuxnet left off.

Chris Glyer gave an excellent presentation in 2014 describing the (limited) forensic artifacts left behind by WMI Event Consumers.[3] The Sysinternals tool Autoruns and the Kansa PowerShell framework both identify WMI event filters and consumers.[4] The PowerShell cmdlet "Get-WmiObject" can also be used as a native means to identify and help remove suspicious entries.

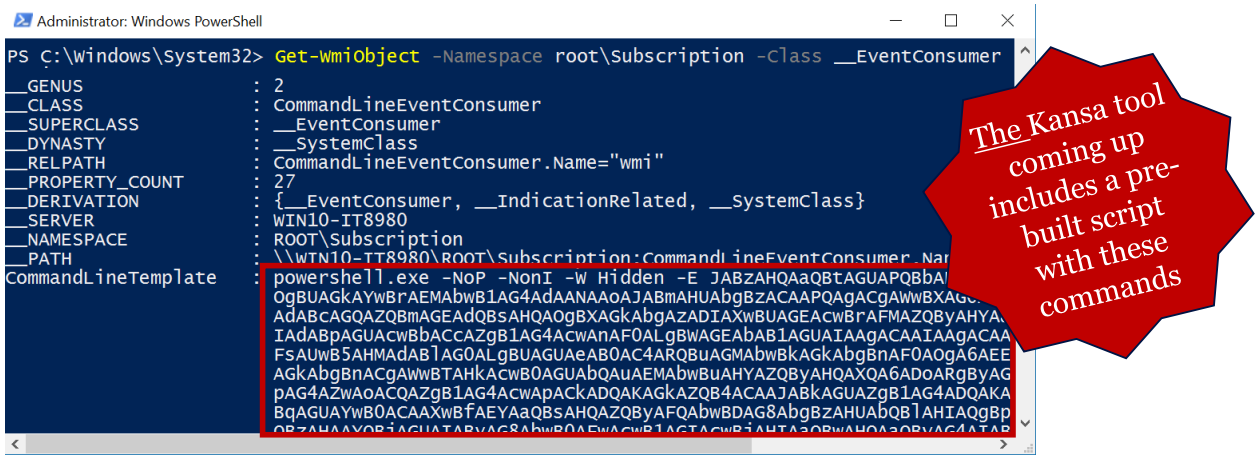
The contents of a sample malicious MOF file follows:

```
#PRAGMA AUTORECOVER
#PRAGMA NAMESPACE("\\\\.\\root\\subscription")

instance of __FilterToConsumerBinding {
    Filter = $Filter;
    Consumer = $Consumer;
};

instance of __EventFilter as $Filter {
 EventNamespace= "ROOT\subscription";
 Name = "GenericFilter";
 QueryLanguage = "WQL";
 Query =
"SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA
'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 200 AND
TargetInstance.SystemUpTime < 320";
};

instance of CommandLineEventConsumer as $Consumer {
   Name = "GenericConsumer";
   RunInteractively=false;
   CommandLineTemplate="C:\Windows\evil.exe";
};
```

[1] Monitoring Events: http://for508.com/2jh-f
[2] Stuxnet Under the Microscope (PDF): http://for508.com/lj9em
[3] There's Something About WMI (PDF): http://for508.com/kc36x
[4] Kansa and WMI Event Consumers: http://for508.com/x36i8

Using PowerShell to Discover Suspicious WMI Events

Get-WMIObject -Namespace root\Subscription -Class __EventFilter
Get-WMIObject -Namespace root\Subscription -Class __EventConsumer
Get-WMIObject -Namespace root\Subscription -Class __FilterToConsumerBinding



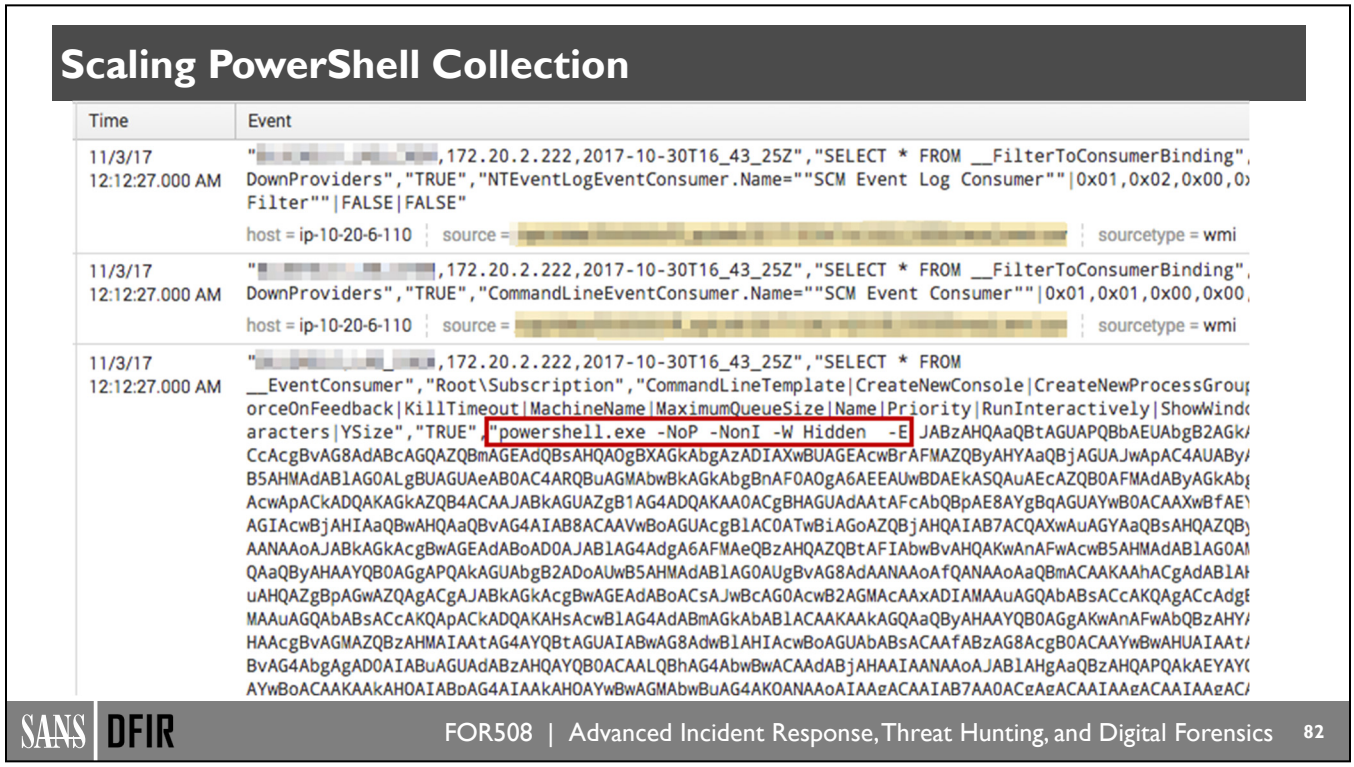
While WMI and PowerShell can be used for attacks, they equally can be used for defense. Native support for WMI and easy scalability makes PowerShell an obvious choice for detecting attacks like WMI event consumers. This is great news because you do not need any fancy tools for detection of one of the more insidious WMI threats. The commands seen on this slide collect the WMI event filters, consumers, and bindings on a system. You can be more specific by using the class parameter (e.g., **`-Class CommandLineEventConsumer`**), but that is only recommended if you are unable to allowlist the standard false positives. You may also consider running the command twice—once as you see on this slide, querying the "root/Subscription" namespace, and once again for the "root/Default" namespace. Microsoft documentation and most of the event consumers seen in the wild use the standard "root/Subscription" namespace, but it is possible to accomplish the same evil persistence by embedded in the "root/Default" namespace. As organizations get better at finding the standard consumers, attackers may move to non-standard locations. Knowing this, the following set of commands would be a best practice:

Get-WMIObject -Namespace root\Subscription -Class __EventFilter
Get-WMIObject -Namespace root\Subscription -Class __EventConsumer
Get-WMIObject -Namespace root\Subscription -Class __FilterToConsumerBinding

and

Get-WMIObject -Namespace root\Default -Class __EventFilter
Get-WMIObject -Namespace root\Default -Class __EventConsumer
Get-WMIObject -Namespace root\Default -Class __FilterToConsumerBinding

The example on the slide shows a clearly bad consumer, with an encoded PowerShell script as the payload. Also note that the Kansa IR collection framework has pre-built PowerShell scripts to collect this information at scale (including collection from both the Subscription and Default namespaces).



While WMI event consumers can be audited on a single system via PowerShell, a more likely scenario is collecting the information from hundreds or thousands of systems using something like PowerShell remoting. That data could then be placed into a database (an ELK stack[1] or Splunk are two good options) where it could be hunted through for anomalies. There are legitimate use cases for WMI event consumers, and nearly every system has some (the "SCM Event Log Consumer" seen on this slide is one common example). However, those legitimate consumers are small in number and predictable, making them easy to allowlist. Event consumers like the one seen on this slide should stick out, even among thousands of systems, if only because "powershell" is a great filter term to find evil in this type of dataset.

[1] ELK Stack: Elasticsearch, Logstash, Kibana – http://for508.com/q1r94

Hunting Notes: WMI Persistence

- Focus on Consumers (CommandLine and ActiveScript)
  - Later, the Event Filter (trigger event) can be determined
- Interesting search terms:

.exe	**.vbs**	**.ps1**
.dll	**.eval**	**ActiveXObject**
powershell	**CommandLineTemplate**	**ScriptText**

- Common false positives:

SCM Event Log Consumer	**BVTFilter**	**TSlogonEvents.vbs**
TSLogonFilter	**RAevent.vbs**	**RmAssistEventFilter**
KernCap.vbs	**NTEventLogConsumer**	**WSCEAA.exe (Dell)**

Note: Be careful when global allowlisting. Attacker consumers named "SCM Event Consumer" seen in the wild

When hunting WMI event data in the enterprise, you will need to develop an allow-list and filters to facilitate anomaly detection in large datasets. If you are collecting this data, there are some excellent opportunities for big wins here. First, focus on event consumers (as opposed to event filters and bindings). The consumers typically have far more interesting data to facilitate high fidelity searching on blocklist terms.

Once you identify a bad consumer, you can then use the name of the consumer to find the binding and associated event filter for the consumer. While not always necessary, the event filter can give more insight into what was used to trigger the execution of that consumer (every five minutes, at boot, whenever a specific user logged on, etc.)

Once you collect some data at scale within the organization, building an allowlist of common normal consumers should be straightforward. Each environment will be different, but this slide lists some of the most frequent legitimate consumers found. A word of caution! From time to time, audit your allowlist to make sure it is not too permissive. As an example, the BadRabbit ransomware variant named their evil event consumer "SCM Event Consumer" to try to blend in (and potentially take advantage of any allowlisting). The normal legitimate consumer with this name is slightly different: "SCM Event **Log** Consumer." Other ransomware variants have used "SCM Event**s** Log Consumer. There have also been reports of the legitimate (and very common) KernCap.vbs script overwritten with a malicious script with the same name. It is best to keep in mind that attackers will try to use our tools and techniques against us!

Notice that this slide calls out two specific types of consumers on which to focus. When investigating WMI event consumers, you may encounter several different types of consumers.[1] To further reduce our data set, we can focus on two used by all of the current attacks seen in the wild: CommandLineEventConsumers and ActiveScriptEventConsumers.

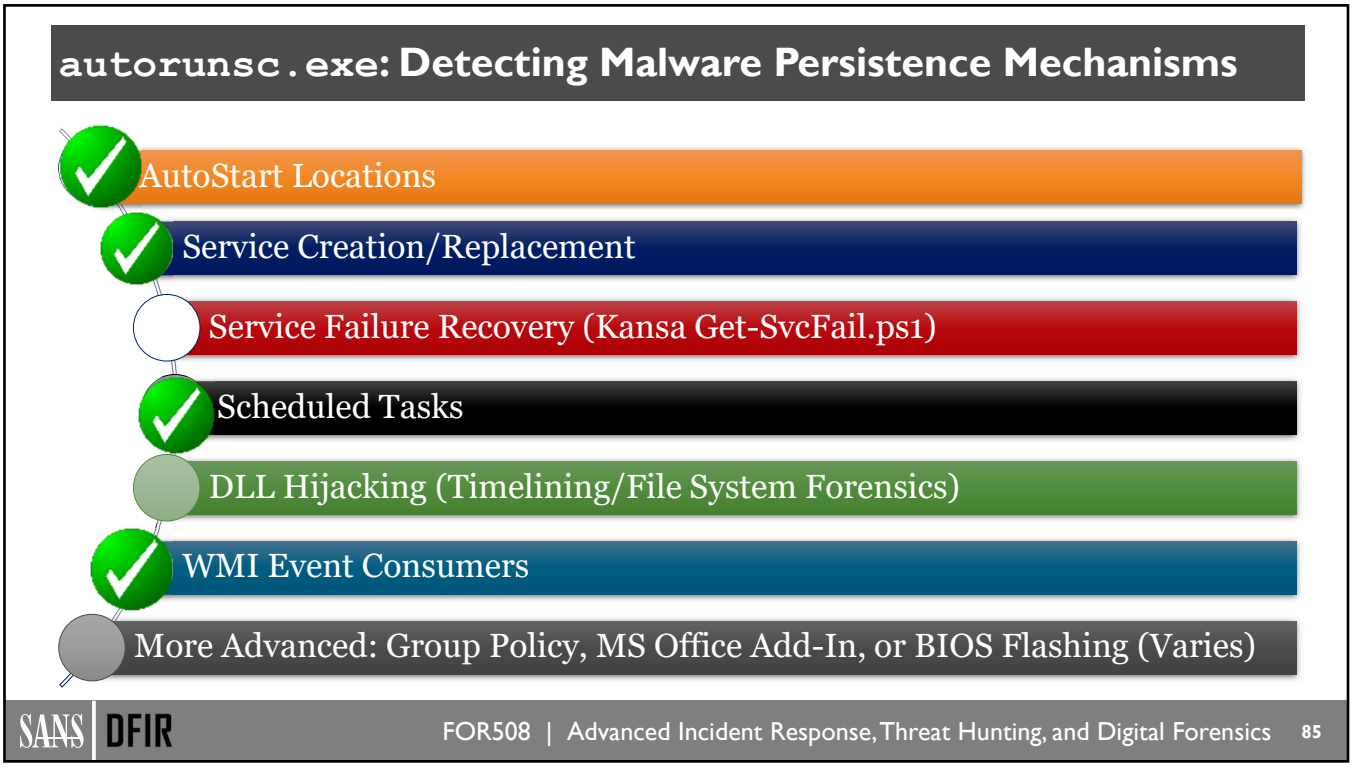
CommandLineEventConsumers allow the payload of an event filter (trigger) to be an executable. You may see a malicious executable, such as evil.exe, in the properties, or something more complex like rundll32.exe or powershell.exe along with parameters.

ActiveScriptEventConsumers are the second common vector for evil event consumers. They are incredibly flexible, using either a path to a script on disk or simply script text. Visual Basic or JScript are the two supported scripting languages for these types of consumers. You will not see PowerShell scripts in this type of consumer.

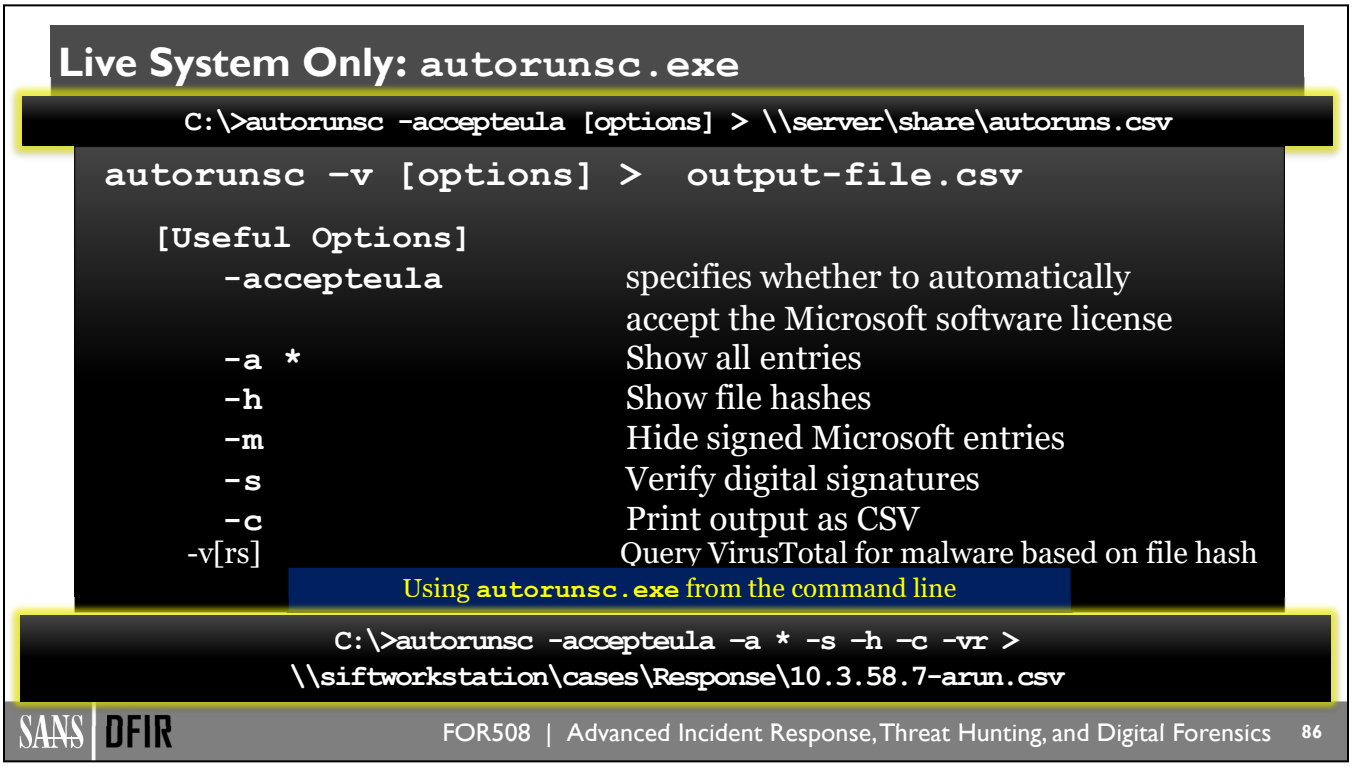
The capabilities of the other possible event consumers can be found in the following table:

ActiveScriptEventConsumer	Execute a predefined script: VB or JScript
CommandLineEventConsumer	Launch an arbitrary process
LogFileEventConsumer	Write to a text log file
NTEventLogEventConsumer	Log a message to Event Log
SMTPEventConsumer	Email a message via SMTP
Custom	Requires custom COM object

[1] Standard Consumer Classes | Microsoft Docs: http://for508.com/9oah4



Autorunsc.exe: Detecting Malware Persistence Mechanisms
Similar to application vulnerabilities, we will likely never reach the end of new persistence mechanism discoveries. The top six items on this list comprise probably 98% of those you are likely to find, with the other 2% being rare and esoteric examples of firmware implants, new forms of DLL hijacking, etc. Although some persistence mechanisms are most often identified via timelining and disk forensics (DLL Hijacking and Group Policy scripts come to mind), the Autoruns tool from Sysinternals has the ability to collect data from the vast majority of other ASEPs. It is a go-to tool for incident responders and is often one of the first items reviewed in an investigation.

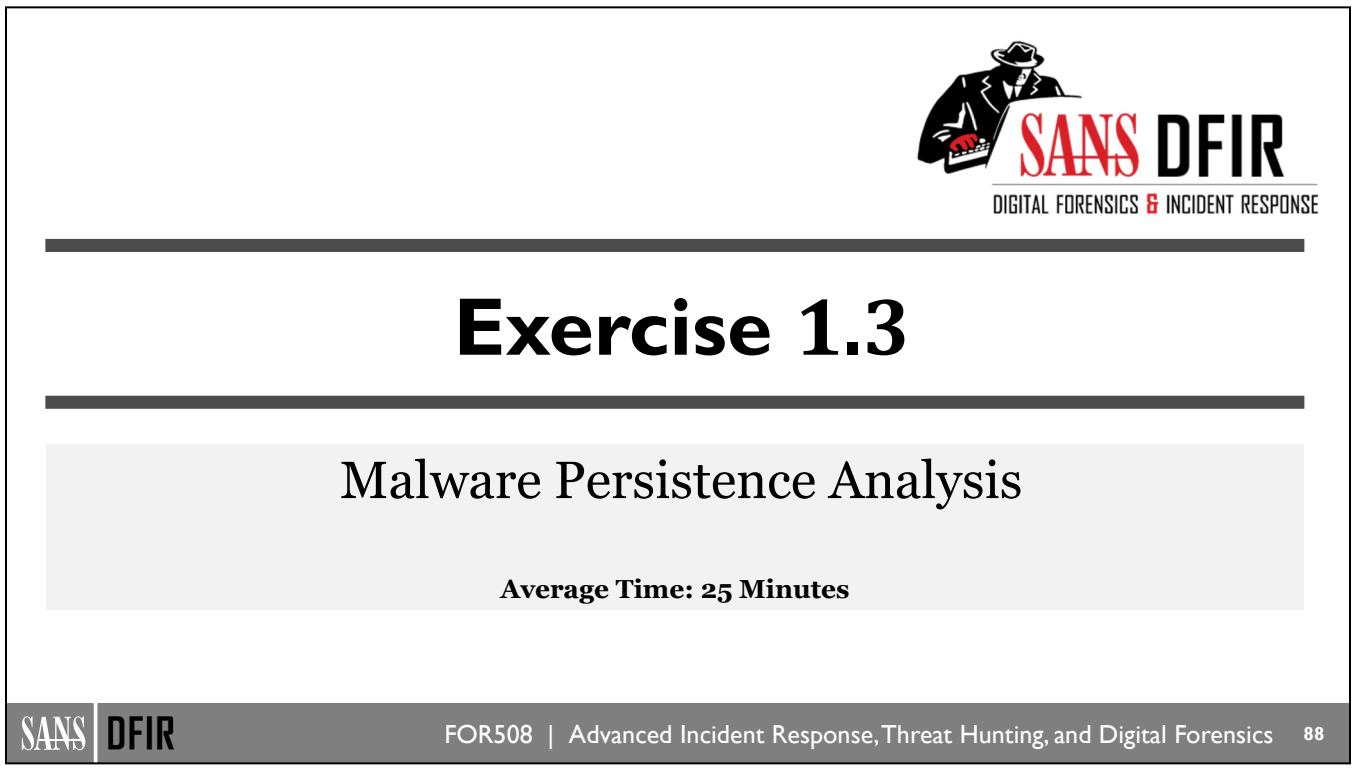


From the Sysinternals tool page for `autorunsc.exe`:
This utility, which has the most comprehensive knowledge of autostarting locations of any startup monitor, shows you what programs are configured to run during system bootup or login, and when you start various built-in Windows applications like Internet Explorer, Explorer, and media players. These programs and drivers include ones in your startup folder, Run, RunOnce, and other Registry keys. Autoruns reports Explorer shell extensions, toolbars, browser helper objects, Winlogon notifications, autostart services, and much more. Autoruns goes way beyond other autostart utilities.

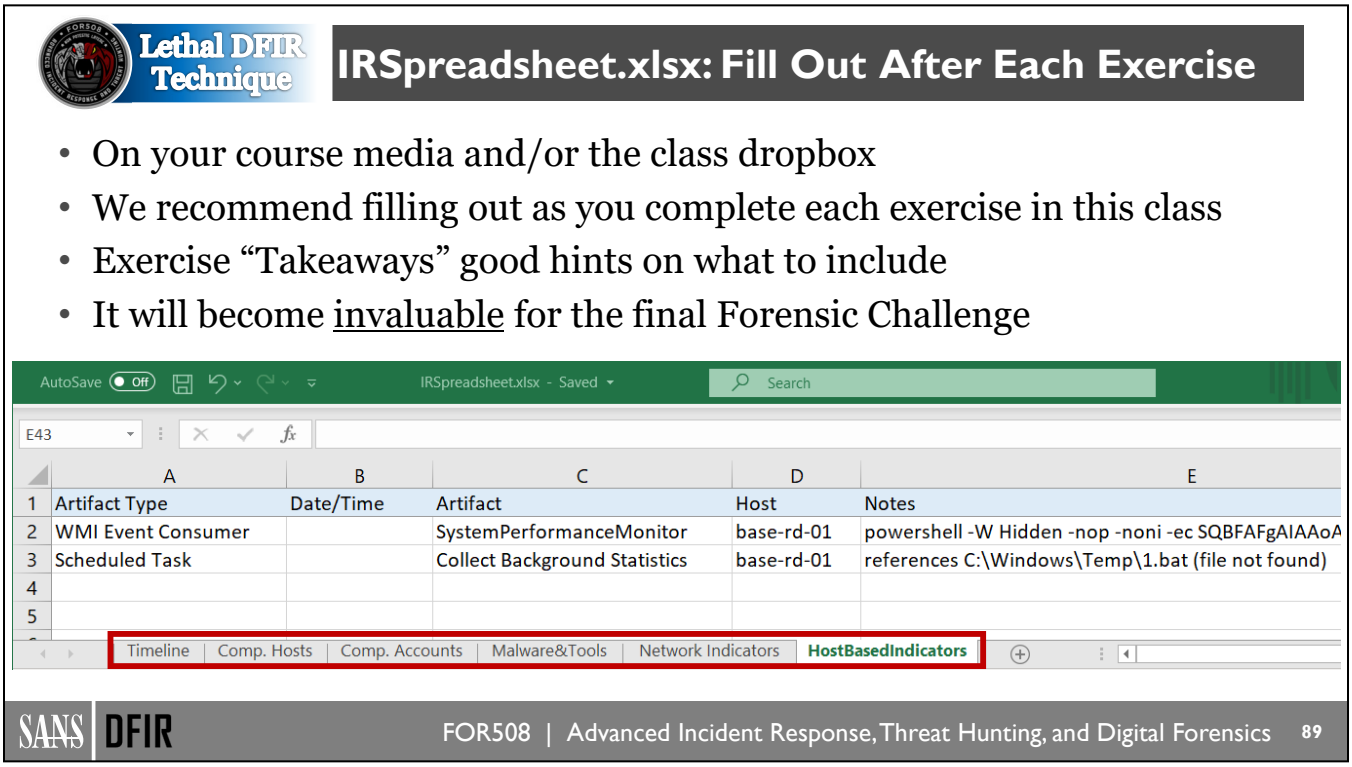
Usage
Autorunsc is the command line version of Autoruns. Its syntax is:

Usage: autorunsc [-a <*|bdeghiklmoprsw>] [-c|-ct] [-h] [-m] [-s] [-u] [-vt] [[-z <systemroot> <userprofile>] | [user]]]
-a Autostart entry selection:
* All.
b Boot execute.
c Codecs.
d Appinit DLLs.
e Explorer add-ons.
g Sidebar gadgets (Vista and higher)
h Image hijacks.
i Internet Explorer add-ons.
k Known DLLs.
l Logon startups (this is the default).
m WMI entries.

```
   n    Winsock protocol and network providers.
   o    Office add-ins.
   p    Printer monitor DLLs.
   r    LSA security providers.
   s    Autostart services and non-disabled drivers.
   t    Scheduled tasks.
   w    Winlogon entries.
-c     Print output as CSV.
-ct    Print output as tab-delimited values.
-h     Show file hashes.
-m     Hide Microsoft entries (signed entries if used with -s).
-s     Verify digital signatures.
-t     Show timestamps in normalized UTC (YYYYMMDD-hhmmss).
-u     If VirusTotal check is enabled, show files that are unknown
       by VirusTotal or have non-zero detection; otherwise, show only
       unsigned files.
-x     Print output as XML.
-v[rs] Query VirusTotal (www.virustotal.com) for malware based on file hash.
       Add 'r' to open reports for files with non-zero detection. Files
       reported as not previously scanned will be uploaded to VirusTotal
       if the 's' option is specified. Note scan results may not be
       available for five or more minutes.
-vt    Before using VirusTotal features, you must accept
       VirusTotal terms of service. See:

           https://www.virustotal.com/en/about/terms-of-service/

       If you haven't accepted the terms and you omit this
       option, you will be interactively prompted.
-z     Specifies the offline Windows system to scan.
user   Specifies the name of the user account for which
       autorun items will be shown. Specify '*' to scan
       all user profiles.
-nobanner
       Do not display the startup banner and copyright message.
```



This page intentionally left blank.



One of the first lessons that any incident responder will learn is to "document as you go." Nothing is more valuable than a spreadsheet or organization of findings you have uncovered along the way. We have included a battle-tested indicators spreadsheet to use for this class. We recommend that you attempt to try and fill this out after each exercise. It helps to document as much information as you can about the incident as you go. You will be amazed at how much information you will begin to juggle around even after ten additional minutes of analysis. Taking a moment to write down a specific IP/host or account that is compromised will be critical to keeping it all straight in your head.

This isn't a requirement when you are just beginning to learn analysis, but it will greatly help on your final challenge to document key items you find now. Identifying evil on new systems is infinitely easier when you already have an initial list of tradecraft to investigate.

Advanced Incident Response and Threat Hunting Agenda

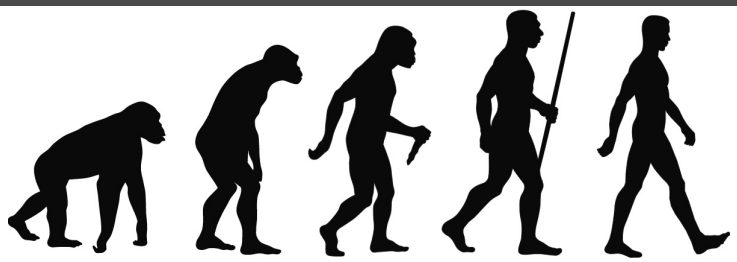
- Incident Response & Threat Hunting
- Threat Intelligence
- Malware-ology
- Malware Persistence
- Incident Response: Hunting Across the Enterprise
- Credential Theft

This page intentionally left blank.

Incident Response: Hunting Across the Enterprise

This page intentionally left blank.

Evolution of IR Scripting



Batch Files: Difficult to deploy, caches credentials, limited feature set, output hard to collect and consume

WMI: Scales easily, safeguards credentials, incredibly granular system reporting, output in XML, HTML, CSV, and so on

PowerShell: WMI + scripting and execution engine

If you have been doing incident response for a while, there is a good chance that you have employed batch scripts to perform live response data collection on systems. Some of you may still lovingly maintain your batch file collection, nicely organized by the Windows operating system type. If that's the case, it is time for an upgrade. Batch files have significant limitations, such as being difficult to deploy at scale (without a third-party tool such as a patch manager), having a limited feature set without the inclusion of a plethora of third-party tools such as the Sysinternals Suite, the inconsistent data output formats, the lack of a built-in capability to return remote data (network shares are often used), and perhaps worst of all, the fact that most implementations dangerously cache the credentials on the remote (and possibly hacked) system. In modern Windows systems, there are far better options.

Windows Management Instrumentation (WMI) has been in place and improved upon since Windows 2000. It was designed for administrative data collection of both local and remote systems. As such, it scales easily (though it will require a VB, PowerShell, or equivalent script to do so), allows detailed querying of a vast number of objects, classes, and properties throughout the OS, uses network (non-interactive) logons to authenticate so that credentials and tokens are not cached on the remote system, and provides a robust output capability including support for XML, HTML, and CSV formats that can be ingested into a database.[1]

Although WMI provides the mechanism for pulling data from the enterprise, PowerShell provides the scripting capability to leverage WMI. The two are inextricably linked and many of PowerShell's built-in cmdlets are simple wrappers around WMI commands. It also extends WMI's capabilities bringing in .NET capabilities, easy output post-processing, and the capability to execute arbitrary binaries to extend your collection capabilities. PowerShell and WMI are nearly the best-case scenario for Windows IR collection. In this section, we show you how to get started.

[1] Running WMI Scripts Against Multiple Computers: http://for508.com/0kx18

Incident Response Using WMIC

- Windows Management Instrumentation CLI:
  - `/node:<remote-IP>` | `/user:<admin acct>`
- Get auto-start processes:

```
wmic /node:10.1.1.1 startup list full
```

- Remote process list:

```
wmic /node:10.1.1.1 process get
```

- Network configuration:

```
wmic /node:10.1.1.1 nicconfig get
```

The PoSh-R2 project automates WMI collection

Windows Management Instrumentation Command Line Interface (WMIC) is available on all Windows systems post Win2000. It gives deep insight into a wide array of operating system information and can be easily scripted via VBScript or the newer PowerShell. WMIC allows remote queries, extending your ability to gather information across the network, so you don't have to be at the system (or at an interactive command prompt). It also provides a safe means to get a majority of our incident response data over the network. As Mike Pilkington described in his SANS Forensic Blog post, unlike PsExec, WMIC uses the native authentication mechanisms, eliminating some of the concern of exposing critical credentials.[1]

The vast array of options in WMIC makes getting started a bit daunting. The first command you should learn is "wmic /?." There are some terrific resources available online regarding using WMIC for incident response. A good starting point is the SANS ISC blog.[2] As an example of the power and simplicity of WMIC, the ISC blog provides the following WMIC command to spot executable running from strange locations:

```
wmic PROCESS WHERE "NOT ExecutablePath LIKE '%Windows%'" GET ExecutablePath
```

WMIC is trivial to script, making it ideal for including in incident response scripts. For example, the PoSh-R2 project on GitHub wraps WMI collection into a PowerShell script.[3] The script can be run on a local system or remotely against specified targets. Output from the script is stored in a collection of CSV and SQLite files. If you want to learn more about WMI commands, you can audit the WMI calls made within the PosH-R2 script by searching for "Get-WMIObject".

[1] WMIC for incident response: http://for508.com/4mlpv
[2] Like a Kid in a WMIC Candy Store: http://for508.com/y2wlf
[3] PoSh-R2 WMI collection script: https://for508.com/obxu8

The Future of IR: PowerShell

- The future of Windows administration
- Default in Win7-11 and Windows Server 2008+
- Powerful scripting language
- WMI, .NET, and COM all in one
- Allows for remote analysis and local logging
- Flexible post-processing and filtering

"PowerShell is both a scripting language and a powerful interactive command interface similar to Bash in UNIX. PowerShell console, where the commands are run, is similar to the Windows command interface, cmd.exe. PowerShell commands can be run in the background or interactively if a particular country's privacy policy enforces an organization to do so. PowerShell V2 is installed by default on Windows 7 operating system."

"PowerShell commands or cmdlets are based on .NET Framework objects, which means that the objects carry multiple aspects or properties of the command. These cmdlets let you access the filesystem and other Windows operating system data stores, such as the registry. PowerShell also provides access to Windows Management Instrumentation (WMI), which means that all the WMI commands that incident responders and information security professionals are familiar with can be run using PowerShell."

"Windows 7 operating system provides an option to run the PowerShell scripts remotely. Microsoft uses the industry-standard WS-Management Protocol to provide remote management features. This comes as a service in Windows 7, which can be enabled either through command line or through group policy."

Sourced from "Live Response Using PowerShell," by Sajeev Nair: http://for508.com/1nsjz

PowerShell Basics

- Verb-noun naming scheme
  - **`Get-Process`**
  - **`Get-Service`**
- Large number of aliases (Dir == Get-ChildItem)
  - **`Get-Alias *`**
- Output is objects that are passed to other cmdlets
  - **`Get-Service | Out-GridView`**
  - **`Netstat.exe | Select-String 'Established'`**
- Direct access to providers like disk and registry
  - **`Get-ChildItem HKLM:Software | Format-Wide`**

If you perform incident response in the Windows enterprise, the time has come to learn PowerShell (PS). Microsoft continues to double down on support for it and a vast amount of the operating system is now being implemented as PowerShell cmdlets. Luckily, it is one of the easiest scripting languages to learn. Cmdlets (small programs or commands) are the building blocks and use a verb-noun naming scheme with the noun being an object or class of objects to do something to. If you are already familiar with the Windows command line interface, you will be pleasantly surprised that nearly all the commands you know have been ported over to PS. This is accomplished through aliases, and there are more than 150 prebuilt aliases. (You can also add your own.) For example, "Dir" is aliased to the cmdlet "Get-ChildItem." You will also notice that native commands and executables can be run from PowerShell just as they were in cmd.exe.

One important concept in PowerShell is that command output is not in strings, as it may look in the terminal. Data is encapsulated in objects, which can then be easily passed to other cmdlets to perform additional processing. One cmdlet might generate a list of objects representing files, computers, services, and network objects; the next in the pipeline might filter and pass on just the ones that are of interest; and the next might call methods to perform actions on the objects. A simple example is output formatting. Output can be easily changed by piping one cmdlet to an outputter such as "Out-GridView," which provides a GUI interface into the data. To display the various components of an object, pipe the command into "Get-Member."

A final important concept is the idea of a provider. PowerShell abstracts collections of items into containers called providers. Providers can be file volumes, the registry, Active Directory, and so on. This makes it trivial to interact with other objects in Windows. In the example on this slide, "Get-ChildItem HKLM:Software," we list the registry keys below the HKLM/Software hive. You can even mount a network share to a PS provider (using the New-PSDrive cmdlet)!

There are an amazing number of great resources for learning PowerShell. One classic book is *Learn Windows PowerShell in a Month of Lunches* by Don Jones and Jeffery D. Hicks. There are also many great online articles. Some relevant ones follow.

PowerShell for incident response:

- PowerShell: Forensic Oneliners – http://for508.com/-a0jb
- Weekend Scripter: Using PowerShell to Aid in Security Forensics – http://for508.com/ur96m

PowerShell to find evil:

- Use PowerShell to Perform Offline Analysis of Security Logs: http://for508.com/0b3q-
- Use PowerShell to Get File Hashes: http://for508.com/izchu
- Compute MD5 Hashes and Find Changed Files: http://for508.com/sb8u4

PowerShell Basics: Remoting

- WinRM Service required
  - Enabled by default on Server 2012+
- **`Enter-PSSession`**
  - Provides remote shell; Alternative to SSH
- **`Invoke-Command`**
  - Allows concurrent one-to-many command execution
- Filtering accomplished on remote host

One of the most useful features of PowerShell is that starting with version 2 (Win7 and above), it has native support for executing commands on remote systems. This feature is at the crux of why PS is so valuable for incident responders. The Windows Remote Management service is used, which is already enabled in many environments for WMI data collection and event log forwarding and is enabled by default in Server 2012. The transfer protocol is WS-Management (WSMAN), which uses SOAP, XML, and HTTP listeners to pass through packet inspection devices. Even though HTTP is employed for the inbound connection, the data transferred is encrypted and credentials are authenticated via Kerberos.

The cmdlet "Enter-PSSession <computername>" provides an interactive remote shell on the target system. However, unlike RDP or some implementations of PSExec, credentials are not cached on the remote system, providing an excellent solution for quick data gathering from a remote host.[1]

The "Invoke-Command" cmdlet is used to send remote tasks to systems without creating an interactive session.[2] The full suite of PS capabilities is available on the remote system, including the capability to run scripts. A script can be passed via the "-ScriptBlock" parameter, or a script can be copied to the remote system using the "-FilePath" option. Although the ability to run scripts remotely is nice, its true power comes in the built-in capability to extend execution from one-to-many. Instead of needing to implement a loop in the script to iterate through a collection of systems, "Invoke-Command" can natively take an array of computer names as an argument. Now with one command, you can run an extremely complicated script across potentially thousands of systems!

You may be thinking, "How long will it take to run a script on thousands of systems?" The answer is probably a lot less than you think! Jason Hofferle did a comparison of using a WMI command to retrieve the last 20 events from the security log of 100 computers using a loop versus using the concurrent connections allowed by "Invoke-Command."[3] The results were astounding, with "Invoke-Command" completing the task on 100 systems in 15 seconds and the loop taking more than 6 hours! Further, he scaled the PS job to 1,000 systems and it completed in 131 seconds.

The big difference here is that "**`Invoke-Command`**" pushes processing and filtering onto the remote host. The WMI loop had to bring the event log all the way back to the host system where the filtering was accomplished. Pushing processing and filtering to the host is **exactly** what we want when doing large-scale IR collection! Finally, PS also includes the capability to manage remote executions as a background job with the "-AsJob" parameter. This is especially useful when running scripts that take a long time or on many systems. The status of remote jobs can be identified with the "Get-Job" cmdlet.

[1] Power of PowerShell Remoting by Mike Pilkington: https://for508.com/elnhv
[2] Invoke-Command: https://technet.microsoft.com/en-us/library/hh849719.aspx
[3] PowerShell Remoting Performance: http://www.hofferle.com/powershell-remoting-performance/

PowerShell Basics: Authentication

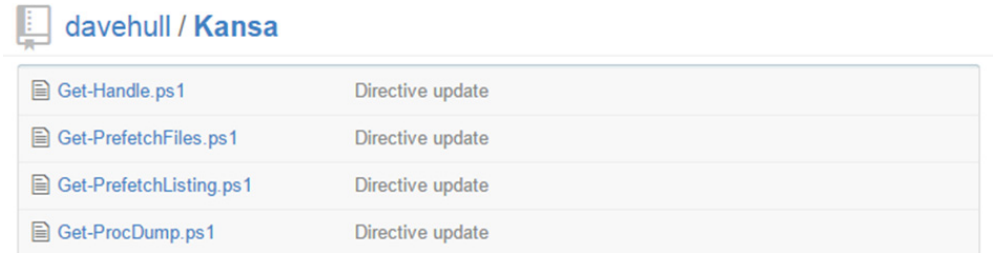
- Best-case scenario from security perspective
- Default is nondelegated Kerberos
  - Precludes delegate token stealing
  - Do NOT use CredSSP (dual-hop) auth
- Non-interactive (Type 3) logon
  - Even **`Enter-PSSession`** does not cache creds
- Credentials not passed to remote system and hence not available to tools, such as Mimikatz, Incognito, etc.

Strong authentication and encryption are taken care of in the background by PowerShell. Remote PS sessions are encrypted, and authentication occurs using Kerberos. Sessions are treated as non-interactive (including Enter-PSSession shell), so password hashes are not cached on the remote system. This makes it far safer to use than something such as RDP. Also, by default, PS uses nondelegated authentication tokens, which means that tokens cannot be stolen from the remote system and then used to authenticate to additional systems. This is the ideal situation from a security perspective but can cause issues if scripts are written in such a way that they require access to a third system. This is called "dual-hop" authentication and is implemented with a service called CredSSP (essentially single sign-on). However, you should aim to NEVER use it. CredSSP will store credentials on the remote system, including the hash and plaintext password extracted by tools such as Mimikatz.[1] In fact, this is one of the reasons why PS is such a great replacement for standard batch scripts. When run on remote hosts, batch scripts often required the host to authenticate to a network share to either pull binaries or store data, effectively requiring that "dual-hop" authentication be used. In general, scripts that pull binaries from shares or dump data to shares require delegation authority, which leaves credentials available on the remote host! By default, PowerShell remoting eliminates these security concerns and ensures that valuable credentials such as Domain Admin are not scattered on systems throughout the environment.

[1] Accidental Sabotage: Beware of CredSSP: http://for508.com/whnby

Kansa: PowerShell IR Framework

- Designed by Dave Hull to scale IR data collections
- Framework organizes data collection and module selection
  - Modules are written as PowerShell scripts
- Easily scales to thousands of systems with new updates
- Not confined to PowerShell cmdlets—execute virtually anything



Kansa uses PowerShell Remoting to run user-contributed modules across hosts in an enterprise to collect data for use during incident response, breach hunts, or for building an environmental baseline.

- If users don't supply their own list of remote systems (targets), Kansa will query Domain Controllers and build the list automatically.
- Error handling and transcription with errors are written to a central file.
- PowerShell remote job management is used, invoking each module on target systems, in parallel, currently using PowerShell's default of 32 systems at a time.
- Output management: As the data comes in from each target, Kansa writes the output to a user-specified folder, one file per target per module.
- A modules.conf file where users can specify which modules they want to run and in what order, lending support to the principal of collecting data in the order of volatility.

Kansa was designed to gather data from hundreds of hosts at a time, given two prerequisites are satisfied: 1) your targets are configured for Windows Remoting (WinRM) and 2) the account you're using has Admin access to the remote hosts (only local admin is required).

Getting a copy of Kansa is easy; simply download it and extract it from GitHub to a location of your choice.[1] The repository contains several folders. The **`.\Modules`** folder contains the plugins that Kansa will invoke on remote hosts. Plugins are aggregated into subfolders by the type of data they are designed to extract. Looking around in this folder will give you a good idea of the current capabilities of the tool.

The **`.\Analysis`** folder contains PowerShell scripts for conducting basic analysis of the collected data. Many of the analysis scripts require **`logparser.exe`**, which is not part of the distro. So, make sure you also download and place logparser.exe in your path. Keep in mind that you can also create your own analysis workflow. Because Kansa output is typically tab-separated, it can easily be imported into the database of your choice; you can run queries against it using Logparser, load it into Excel, and so on.

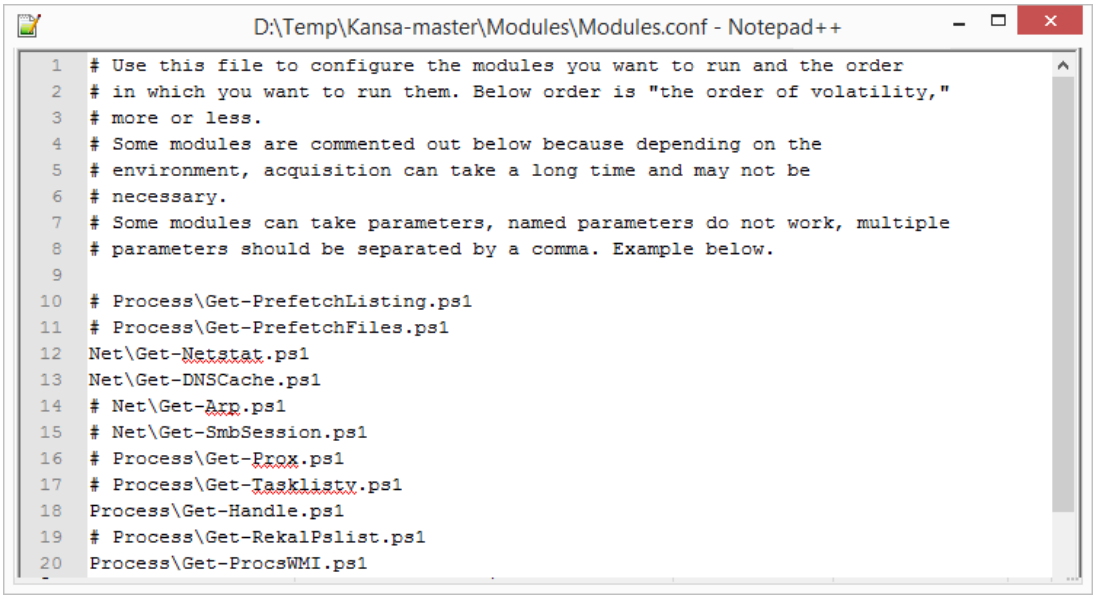
Kansa's capabilities can be easily expanded by writing new modules. Because the language is PowerShell, the barrier to entry is low and the capabilities available are massive. Should you modify or write new plugins, consider submitting them back to the project.

A common criticism of all live response collection is its susceptibility to being fooled by a malicious rootkit on the system. Rootkits are used by malware to hide and often exist at low levels in the system like the operating system kernel. Because most live response tools (including system commands, WMI, .NET, and PowerShell) rely upon the Windows API to collect data, a rootkit can easily subvert those API functions to return incomplete data. However, keep in mind that rootkits are rare and are not used nearly as prevalently as some would suggest. If a rootkit is present, then data collection should be accomplished via memory or disk forensics—both of which can identify a rootkit and its activities (as we will see later in this course). But deep dive forensics is much too time-consuming to run on hundreds of systems. Hence, we typically make the trade-off of speed and efficiency of live response toolkits for large-scale collection while augmenting that analysis with deep dive forensics of a smaller sample of systems. Of course, knowing whether a rootkit is in play and where it might be is a chicken-and-egg problem. That question can often be answered by security alerts, forensic examinations, or via threat intelligence.

Some text used with permission from Dave Hull (http://trustedsignal.blogspot.com/ and https://github.com/davehull/Kansa)

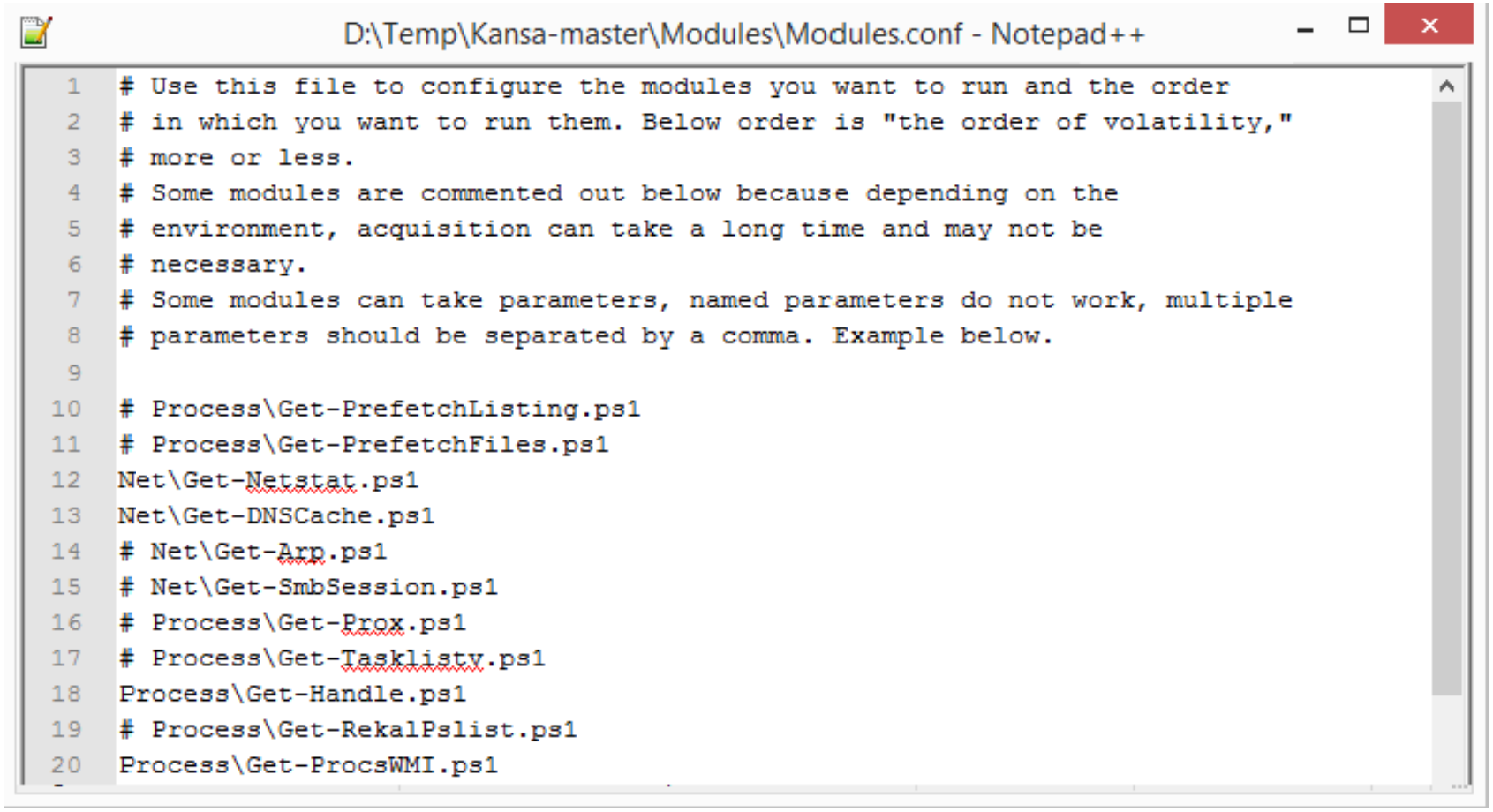
[1] GitHub Kansa: http://for508.com/so5mk

Kansa `Modules.conf`



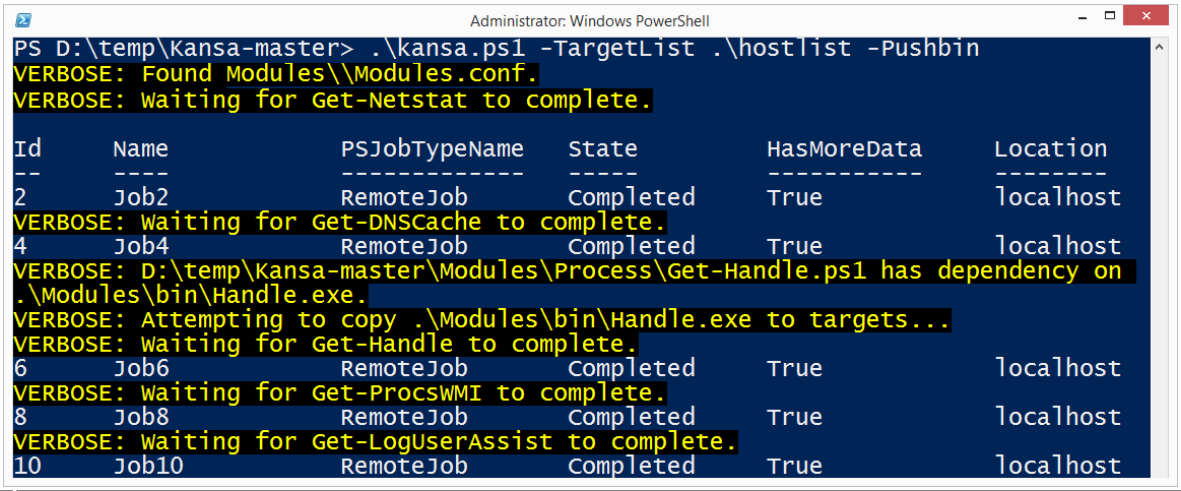
Kansa modules are written in PowerShell and are selected at runtime by referencing the Modules.conf file within the Modules folder. This file is a simple flat configuration file that contains the name and relative location (notice that subfolder names are included with the script names) of each script to be run. As new scripts are written or added to the repository, Modules.conf can be easily updated to request those modules be run. In addition, to prevent a module from running, preface the line with a "#," effectively commenting it out of the file.

The Modules.conf file should be set up to respect the Order of Volatility of data collected. Artifacts that change frequently such as network data, system memory, or Prefetch should be collected early in the process so that evidence is not unnecessarily overwritten. Simply moving those script names closer to the top of the file ensures they are run before others. For example, note that in this slide, we see scripts listed at the top, which are responsible for collecting highly volatile data such as Prefetch, DNScache, and Arp information.



Running Kansa

```
.\kansa.ps1 -OutputPath .\Output\ -TargetList .\hostlist
-TargetCount 250 -Verbose -Pushbin
```



A simple command line for Kansa might look like the following:

```
PS C:\tools\Kansa> .\kansa.ps1 -TargetList .\hostlist -Pushbin
```

Although the .\Modules folder holds all the available scripts to run, **`kansa.ps1`** is the script used for kicking off data collection. The command line arguments are straightforward. "-TargetList .\hostlist" tells **`kansa.ps1`** to run collectors against the systems listed in the hostlist file, which should contain one host per line. If you omit this argument, Kansa will query Active Directory (AD) for the list of computers and target all of them. Querying AD in this way requires this Active Directory module that's bundled with Remote Server Administration Tools, so you'll need that installed if you're not providing a list of targets via -TargetList.[1] If using the AD option, you should also consider including the "-TargetCount" parameter to limit the total number of systems queried (perhaps using a small sample for a test run, and subsequently omitting it to run across the entire environment).

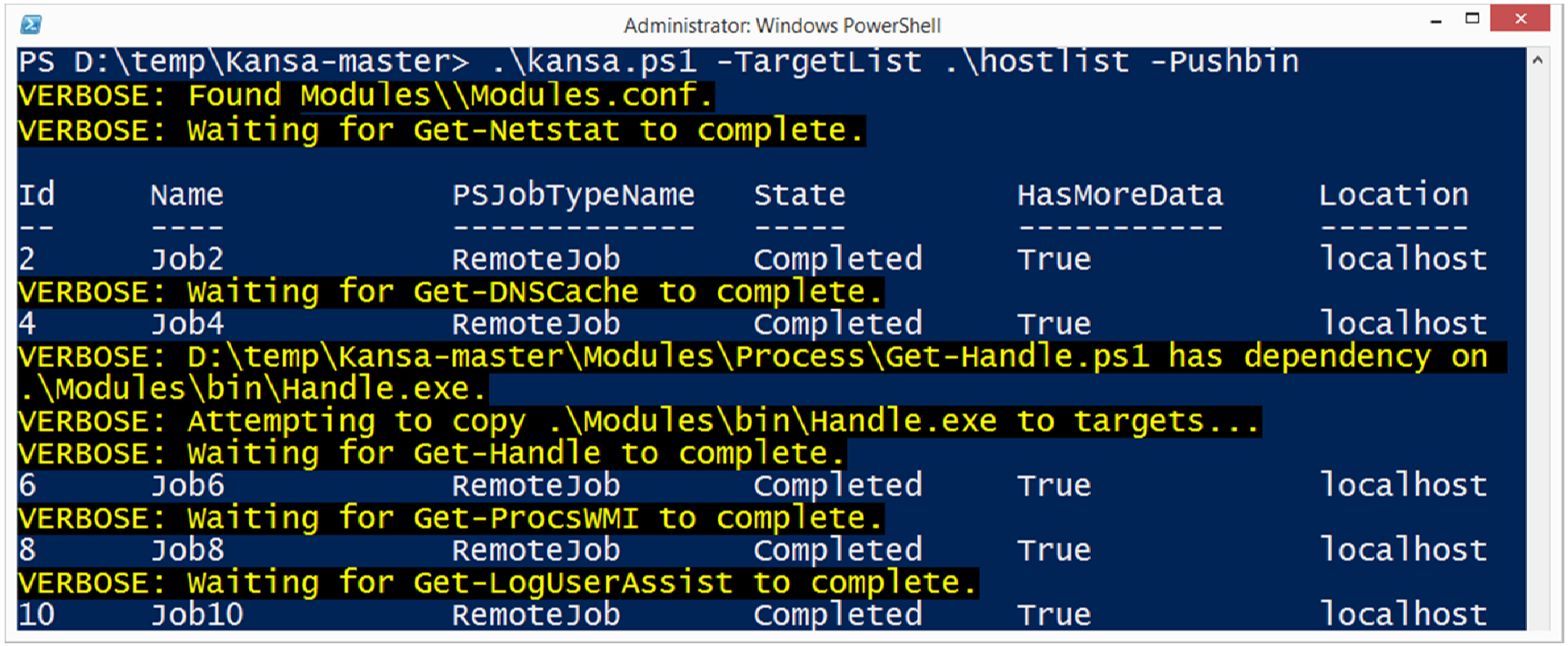
The results returned by each system will be written to an Output folder within the Kansa folder by default. Each collector's output is written to its own folder. Collector scripts are typically written to take the PowerShell objects returned from remote systems and convert them to easy to parse TSV.

A final important option is "-Pushbin," which is required by scripts that employ third-party binaries. If this argument is provided and a script with a binary dependency is run, **`kansa.ps1`** will first copy any required third-party binaries to the targets before running the script. All in all, this is a handy way to run third-party tools remotely!

The "Verbose" option provides additional debugging information to the terminal.

[1] Download Remote Server Administration Tools: http://for508.com/jucow


```
.\kansa.ps1 -OutputPath .\Output\ -TargetList
.\hostlist -TargetCount 250 -Verbose -Pushbin
```



Kansa + Third-Party Tools

- Kansa is not limited to WMI or PS cmdlets
- It can also push and execute binaries
  - Place in the **`.\Modules\bin`** directory
  - Use **`–Pushbin`** argument
  - Remove binaries after execution with **`–Rmbin`**
- Output returned and formatted as PS objects
- Binaries can be deleted or left for future use

Get-Autorunsc.ps1	**Get-FlsBodyfile.ps1**	**Get-ProcDump.ps1**
Get-CertStore.ps1	**Get-Handle.ps1**	**Get-RekalPslist.ps1**

Although PowerShell is extremely powerful, there are certainly instances in which a third-party collection tool is necessary. Luckily, Kansa is not limited to Windows native commands or PowerShell cmdlets. The process to include a third-party binary in a script is simple: Copy the binary into the .\Modules\bin\ directory, include a special comment on the second line of the collector script, and ensure you are running kansa.ps1 with the –Pushbin argument.

The special comment that must be included in the script is referred by the author as a directive. The directive for the `Get-Autorunsc.ps1` module looks like this:

```
# BINDEP .\Modules\bin\Autorunsc.exe
```

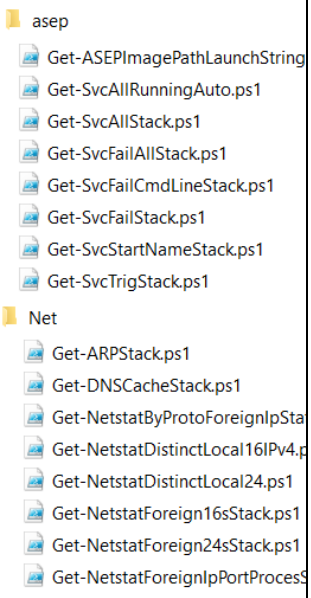
Note that it is an actual comment, but that Kansa knows to look for the "BINDEP," which stands for "binary dependency." This directive tells Kansa to copy .\Modules\bin\Autorunsc.exe to the remote targets before executing the script. By default, binaries will be copied to the %SystemRoot% folder (usually C:\Windows). It is up to the script author as to whether the binary should be deleted after use. One reason that deletion may not be desired is if you plan to run the script again in the future and want to speed up the process. If the binary is already in place on the target system, you can omit the –Pushbin argument. If you would like to remove the binary after use, add the –Rmbin argument.

Several modules currently depend on third-party binaries, including:

- Get-Autorunsc.ps1
- Get-CertStore.ps1
- Get-FlsBodyfile.ps1
- Get-Handle.ps1
- Get-ProcDump.ps1
- Get-RekalPslist.ps1

Kansa Analysis Scripts

- Kansa can also pre-filter and organize data
- Scripts are located in the .\Analysis folder
- Most scripts use "stacking"
  - Least frequency of occurrence analysis
  - Generic stacking script: **`Get-LogparserStack.ps1`**
- "Meta" scripts look at indicators like file size
  - Identification of output file size deviations
- The **`-analysis`** flag auto-runs scripts within **`Analysis.conf`**



Although Kansa is a data collection framework, Dave Hull had the foresight to plan for it to also facilitate analysis. Imagine that you just did a Kansa run of a 1,000 systems. How would you begin to analyze that data? A database or tool such as Splunk comes to mind, but because the data is in simple text format, PowerShell can also be used for parsing. The **`.\Analysis`** folder of Kansa holds the current set of scripts to filter and organize data to facilitate analysis and "finding evil."

The current set of analysis scripts largely utilize a technique called "stacking" or "least frequency of occurrence." The idea is simple but powerful. Malicious activity should be rare in the environment. Whether it is a malicious DLL, network port, or domain name, malicious items should be rare and infrequent across your systems. By stacking all of the data across many systems and counting up unique occurrences, rare items rise to the top of the list. Of my 1,200 workstations, why do I only have four with this specific autorun registry key? Why does this one server have a service that is not seen anywhere else in my network? Why is this workstation communicating with a server that no other system is connecting to? The stacking scripts largely rely upon the free Log Parser tool to do the parsing and frequency counting. The results are then exported to a new file, ready to be reviewed by an analyst.

While Kansa analysis scripts provide a powerful capability, they are really just a start and they provide excellent examples for extending the scripts for other purposes. You might notice a script named **`Get-LogparserStack.ps1`**. This is a generic script that is designed to stack any content, assuming the output format is the same. Upon running the script, the analyst is prompted to provide specific columns to collect and "stack." Your team could easily have a hundred or more scripts, each designed to find one type of anomaly or signature. Further, that set of scripts could then be set up to run every time you execute Kansa. The **`analysis.conf`** file works similar to the **`modules.conf`** file with the difference of it containing the set of analysis scripts to run after data collection has completed. The –analysis flag tells Kansa to look for **`analysis.conf`** to make all of this happen.

Kansa Analysis Script Example

```
if (Get-Command logparser.exe) {
    $lpquery = @"
    SELECT
        COUNT(ImagePath, LaunchString, MD5) as ct,
        ImagePath,
        LaunchString,
        MD5,
        Publisher
    FROM
        *autorunsc.tsv
    WHERE
        Publisher not like '(Verified)%' and
        (ImagePath not like 'File not found%')
    GROUP BY
        Imag
        Laur
        MD5,
        Publ
    ORDER BY
        ct A
```

LogParser command to stack unsigned Autoruns output from multiple Kansa output files

Results from 10 domain controllers shown below (note: no outliers)

ct	Image Path	MD5
10	c:\windows\system32\cpqnimgt\cpqnimgt.exe	78af816051e512844aa98f23fa9e9ab5
10	c:\hp\hpsmh\data\cgi-bin\vcagent\vcagent.exe	54879ccbd9bd262f20b58f79cf539b3f
10	c:\windows\system32\cpqmgmt\cqmgstor\cqmgstor.exe	60668a25cfa2f1882bee8cf2ecc1b897
10	c:\program files\hpwbem\storage\service\hpwmistor.exe	202274cb14edaee27862c6ebce3128d8
10	c:\hp\hpsmh\bin\smhstart.exe	5c74c7c4dc9f78255cae78cd9bf7da63
10	c:\msnipak\win2012sp0\asr\configureasr.vbs	197a28adb0b404fed01e9b67568a8b5e
10	c:\program files\hp\cissesrv\cissesrv.exe	bf68a382c43a5721eef03ff45faece4a

Many of the current Kansa analysis scripts rely upon the free Microsoft tool Log Parser. These scripts expect logparser.exe to be in the system path so that it can find it.

On this slide, we see part of the Get-ASEPImagePathLaunchStringMD5UnsignedStack.ps1 script. The SQL-like language you see with "SELECT" and similar commands is the SQL input that Log Parser expects. Log Parser is a powerful tool and can be learned relatively quickly.[1] In this example, it was used to perform frequency counting of unsigned binaries from the Autoruns output provided by running .\Modules\ASEP\Get-autorunsc.ps1 across ten domain controllers. An analyst would be looking for outliers, such as a binary that was referenced on only one or two of the domain controllers. In this case, all ten of the domain controllers had the exact same set of binaries. (Each has a value of 10 in the "ct" column.) If any were suspicious, the next step would be to cross-reference the provided MD5 hashes with a known-goods database.

[1] Getting Started with LogParser: http://for508.com/r-idw

```
if (Get-Command logparser.exe) {
    $lpquery = @"
    SELECT
        COUNT(ImagePath, LaunchString, MD5) as ct,
        ImagePath,
        LaunchString,
        MD5,
        Publisher
    FROM
        *autorunsc.tsv
    WHERE
        Publisher not like '(Verified)%' and
        (ImagePath not like 'File not found%')
```

LogParser command to stack unsigned Autoruns output from multiple kansa output files

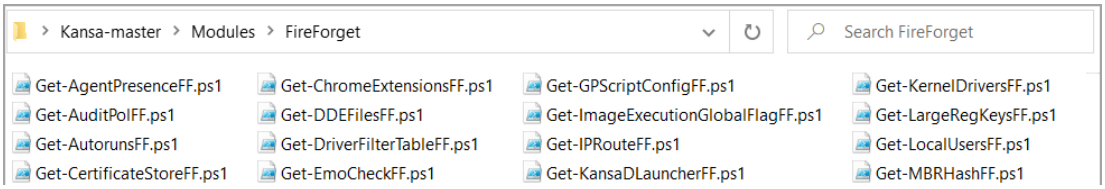
Results from 10 domain controllers shown below (note: no outliers)

ct	Image Path	MD5
10	c:\windows\system32\cpqnimgt\cpqnimgt.exe	78af816051e512844aa98f23fa9e9ab5
10	c:\hp\hpsmh\data\cgi-bin\vcagent\vcagent.exe	54879ccbd9bd262f20b58f79cf539b3f
10	c:\windows\system32\cpqmgmt\cqmgstor\cqmgstor.exe	60668a25cfa2f1882bee8cf2ecc1b897
10	c:\program files\hpwbem\storage\service\hpwmistor.exe	202274cb14edaee27862c6ebce3128d8
10	c:\hp\hpsmh\bin\smhstart.exe	5c74c7c4dc9f78255cae78cd9bf7da63
10	c:\msnipak\win2012sp0\asr\configureasr.vbs	197a28adb0b404fed01e9b67568a8b5e
10	c:\program files\hp\cissesrv\cissesrv.exe	bf68a382c43a5721eef03ff45faece4a

Distributed Kansa + Fire & Forget

```
PS C:\Scripts\kansa> .\DistributedKansa.ps1 -ModulePath .\Modules\FireForget\Get-PowershellVersionFF.ps1 -Credential $cred `
-KansaServers .\kansa_servers.txt -Overwrite -KansaRemotePath "C:\Kansa" -TargetList .\targets.txt -ElkAlert 192.168.0.33 `
-ElkPort 1337 -AutoParse -SafeWord "Ns5(Ksxp98" -FireForget -FFwrapper .\Modules\FireForget\FFwrapper.ps1 `
-FFStagePath .\Modules\FFStaging -FFArgs @{ElkPorts = [array]@(1337); killSwitch = $True; ElkServers = [array]@("192.168.0.33");`
VDIcheck = $False; SafeUser = "HuntUser02"; HuntID = "test";  Notify = $False;  maxDelay = [int]3600;  domain = "CORP.COM"; }
```

- Epic upgrade provided by John Ketchum and USAA Team
- Scale collection from ~150 to 150,000+ systems
  - kansa.ps1 → DistributedKansa.ps1
  - Scripts included to set up distributed Kansa-Servers
  - Fire & Forget modules collect and send data asynchronously to ELK
  - Kill switches, VDI/CPU throttling, alert suppression, and metrics



In perhaps one of the most incredible pull requests in GitHub history, John Ketchum and the threat hunting team from USAA Bank added an unprecedented number of features to Kansa in mid-2020. In a must watch presentation, John provides the history as the bank incrementally added to the project to get around collection bottlenecks, solve security and monitoring issues, and eventually end up with a tool capable of performing complex threat hunting tasks across 150,000 systems.[1] All of their changes have now been accepted into the framework while keeping the original functionality present for backwards compatibility. Kansa is now really two tools in one: a simple tool to run PowerShell collections on a small number of systems at a time (most practitioners have historically chunked target lists into 150 or less systems) and a massive distributed collection platform engineered to work on complex networks of almost any size. You can start using simple Kansa collection, prove the model, and then grow it to meet the needs of a larger enterprise.

The features added are far too numerous to cover in any depth in the time available here, but include the capability to set up distributed Kansa servers to load-balance collection requests, set up servers to host executables to be pulled from the client side (reverse of the -Pushbin option), "fire and forget" modules that run asynchronously and send results directly to the database (skipping the bottleneck of the Kansa application), and formatted output ready for inclusion into an elastic search database. Safety mechanisms include automated helpdesk alerts, staggered execution, a killswitch, VDI abort criteria, CPU limiter, and cleanup scripts to use when an abort is necessary. Mr. Ketchum also demonstrates their additions of EDR "safe words", access to a password vault API, and integration with SOAR platforms to ensure that unusual PowerShell collection doesn't trip alarms and spin up the incident response team.

The code is well documented PowerShell and the slide deck does an excellent job of stepping through the thought process of the various additions to Kansa.[2] The case studies showing real-world use of the new features at the end of the presentation are enlightening and excellent examples of threat hunting performed with home-grown tooling. We owe a debt of thanks to Mr. Ketchum and USAA for sharing their work with the community!

[1] Kansa for Enterprise Scale Threat Hunting w/ Jon Ketchum (video): https://for508.com/95tpx
[2] Kansa for Enterprise Scale Threat Hunting (slides): https://for508.com/7fntg

Exercise 1.4

Kansa Stacking Collection and Analysis

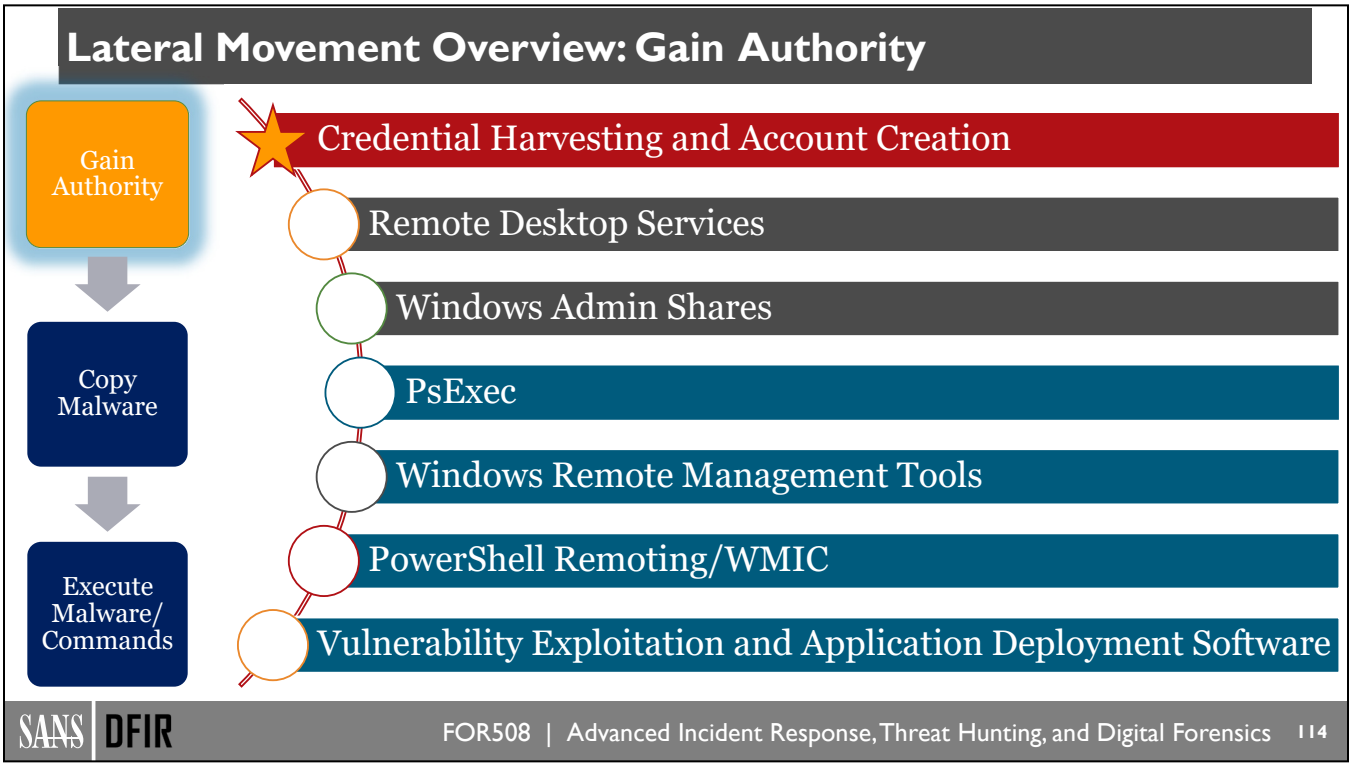
Average Time: 40 Minutes

This page intentionally left blank.

This page intentionally left blank.

Credential Theft

This page intentionally left blank.



When performing incident response, it is important to understand attacker behavior so you can quickly "get to where the hackers are". Lateral movement is essential to the ability to compromise a network and accomplish the attacker's objectives. Understanding the universe of possible lateral movement tools and techniques allows responders to better find and anticipate attacker activity. In this section, we will cover the most common forms of lateral movement.

Credential Harvesting

- Priority #1 post-exploitation
  - Domain admin is ultimate goal
- Nearly everything in Windows is tied to an account
  - Difficult to move without one
- Easy and relatively stealthy means to traverse the network
  - Account limitations are rare
- "Sleeper" accounts can provide access after remediation

Detection

- Event Logs*
  - 4624 Logons
  - 4720 Account creation
  - 4776 Local account auth
  - 4672 Privileged acct usage
- Unix "secure" logs
- Auditing new accounts
- Anomalous logins
  - Workstation to workstation
  - Sensitive networks
- After-hours logins

* Event logs and IDs will be covered in the next section

Attackers prioritize the collection of credentials almost immediately after the post-exploitation phase of the attack cycle. Attackers rarely have the level of access necessary to move freely throughout the enterprise at the time of initial compromise. Privilege escalation or the collection of additional and more privileged credentials is almost always required to fully compromise a network. Many of the lateral movement techniques discussed in this section require legitimate credentials (or elements thereof). Since nearly every activity in Windows is tied to an account, it can be difficult to laterally move without a privileged account (vulnerability exploitation is one of the few exceptions).

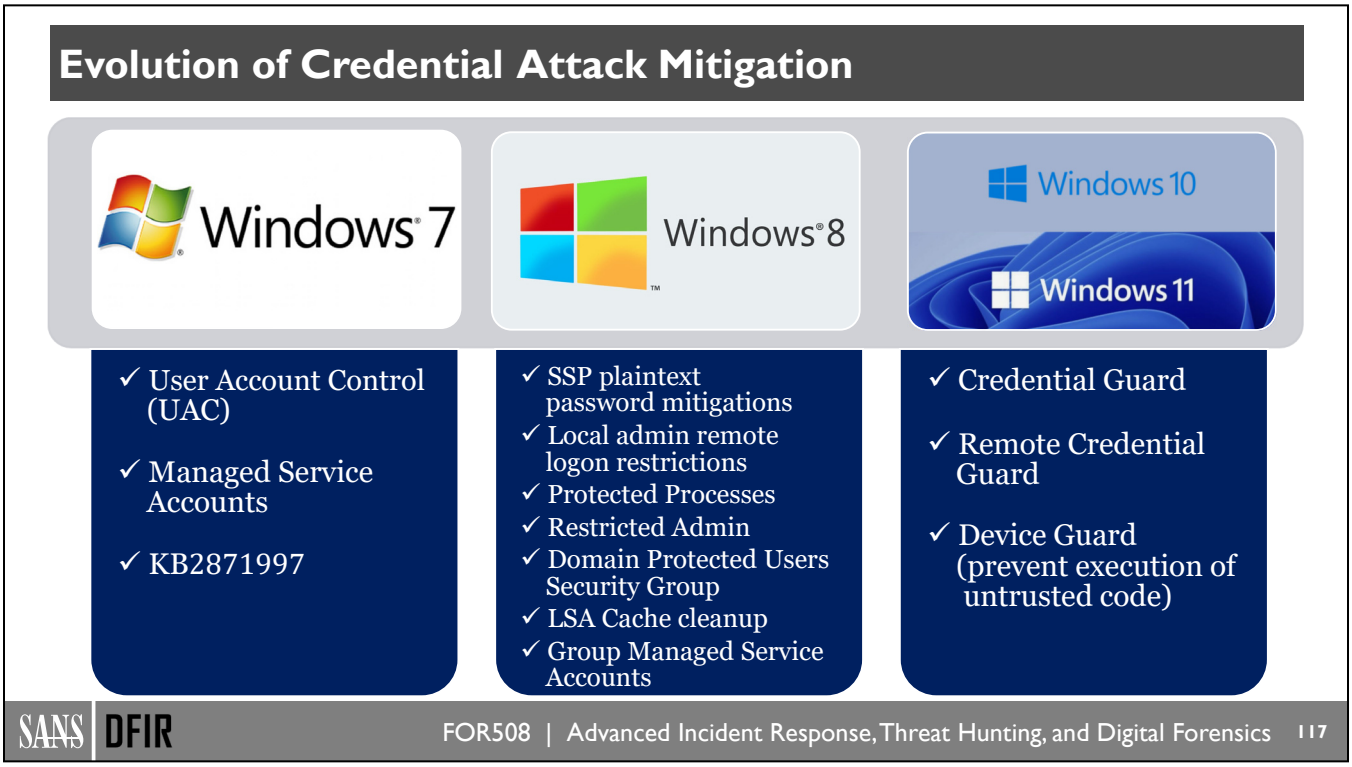
Luckily, account activity is easy to monitor and track. Assuming your enterprise centrally collects and manages logs, tracking account usage is very feasible (especially ultra-privileged accounts like domain admin). Both Windows and Unix provide excellent logging of account activity. In addition to privileged and known compromised account tracking, analysts should also monitor for new account creations. These events occur infrequently and are used by some threat actors to evade monitoring or create "sleeper" accounts that maybe escape remediation efforts and allow immediate access back into the enterprise post-breach. As monitoring and account management efforts become more mature, identifying anomalous logins between devices, network segments, and times of day can be achieved. As an example, a very common attacker behavior is to laterally move between workstations, which is rare in most server-client models. This lateral movement capability can be limited and audited.

A common account vulnerability present in many enterprises is a shared local admin account utilizing the same password across many devices. In this scenario, the compromise of one system allows lateral movement to many others, along with the possibility of collecting more credentials at each additional system accessed. Local account usage is rare in the enterprise and can be easily monitored via event logs. When paired with other account monitoring including privileged account usage, abuse can often be identified. This common vulnerability has not gone unnoticed by Microsoft, and the most recent operating systems have greatly limited the capabilities of a local administrator account. As an example, a local admin account cannot remotely write to

C$ and Admin$ shares and cannot use some remote management tools like schtasks, at, or WMI (Note: if the Windows Remote Management service is enabled, WMI/PowerShell will still work using a local admin account). Please note that these protections do not apply to the built-in admin account, RID 500, which unfortunately many organizations still enable and use.

Finally, enterprises can further reduce their attack surface by creating unique passwords for every local admin account and denying network logons for these accounts.[1]

[1] Technet – Local Accounts: http://for508.com/prc2-



Credential theft problems in Windows should take no one by surprise. It has been an issue since the very beginning. Unfortunately, like many other security issues in Windows, poor initial credential implementation coupled with the ever-present backward compatibility demands has been Microsoft's nemesis in securing the operating system.

Vista/Windows 7

The Windows XP era was a security low point, and several mitigations were baked into the Vista/Windows 7/Server 2008R2 release. Mandatory access control and least privilege support were implemented with a new feature called User Account Control (UAC). Fewer applications required elevated privileges and admin accounts were restricted to user-level permissions by default (for web surfing or file opening).[1] Note that while UAC can be helpful, it was not designed to be a security boundary and does not employ important barriers like process isolation.[2]

Managed Service Accounts were released with Server 2008R2 and are one of the few mitigations available to combat some of the most advanced Kerberos ticket attacks. This feature specifically helps with service accounts being run with domain account rights. It provides both frequent password changes (30 days by default) and long and complex passwords for these accounts.[3] While an excellent idea, the initial implementation was lacking. The latest iteration, Group Managed Service Accounts, is much more flexible and admin-friendly.

Note that KB2871997 nicely backported many of the protections released in Windows 8 to Windows 7/Server 2008R2, but some protections still require an Active Directory functional level set to at least 2012R2.[4]

Windows 8.1

Credential attack mitigations were one of the few things to get excited about during the Windows 8/8.1 release. Microsoft took a hard look at the havoc being caused by tools and techniques like Mimikatz and pass the hash

and decided to act. CredSSP and other single sign-on (SSO) credentials like TsPkg and Wdigest are no longer cached in memory by default. This removes the Mimikatz ability to recover plaintext credentials from these sources. A new security group was added to facilitate restricting local (admin) accounts from network or remote interactive logons to domain-joined systems. This breaks pass-the-hash attacks, mounting remote shares, remote WMI, PsExec, and remote scheduled tasks using local administrator accounts (a favorite technique of attackers who discover a shared local admin account).

The concept of protected processes was introduced in Windows 8. Protected processes can only load signed code and can only be attached to by other protected processes.[5] The LSASS process, in particular, was singled out as protected in order to defeat many of the common credential dumping tools that inject code into LSASS to collect credentials. Unfortunately, protection is off by default and is unlikely to get rolled out widely through the enterprise. Mimikatz gets around this protection via a signed driver.

Two new account protections also debuted in Windows 8. Remote Desktop can now be executed using the "/restrictedAdmin" switch, ensuring that credentials and tickets are not pushed to target systems during interactive RDP sessions (it can also be forced for accounts using Group Policy). This feature was originally on by default on Server 2012 but subsequently changed to off by default due to the unintended consequence of opening up RDP sessions to pass-the-hash attacks. It has been succeeded by Remote Credential Guard in Windows 10+. More important than Restricted Admin is the Domain Protected Users security group. This group is designed to protect high-value (privileged) accounts. Members of the group cannot authenticate via weak (and often abused) NTLM, CredSSP, or Digest Authentication.[6] This defeats some Mimikatz capabilities and pass-the-hash tools. Credentials are not cached on systems and tokens cannot be delegated (removing the threat of cached credentials and token stealing). NT hashes are not cached on remote servers. Finally, Kerberos ticket life is reduced (to four hours) and the weaker RC4 ticket encryption is denied, which is used by multiple ticket-based attacks. Needless to say, the Domain Protected Users Group is one of the most important security features added to this release.

Finally, Windows 8 also improved upon the cleanup of credentials after termination of user sessions and released an improved capability to administrate and protect service accounts named Group Managed Service Accounts.

Windows 10 and 11

With the release of Windows 10, Microsoft improved upon the previous security additions in a few important ways. Credential Guard is a game changing technology that isolates hashes and tickets using a security boundary enforced by machine virtualization.[7] While it has software and hardware requirements and will be some time before it is widely deployed, it effectively defeats all of the current user account hash and ticket dumping techniques currently in the wild. Remote Credential Guard is an update to Restricted Admin and protects any account (not just admin) during RDP sessions. Finally, Device Guard is a new take on application control that can lock down systems to prevent untrusted code (like credential dumping utilities) from running on a system. While very effective, it appears to be challenging to tune and will likely only be found on a limited number of very important systems in the enterprise. Windows 11 inherits the improvements introduced in Windows 10 with no substantive changes.

[1] User Account Control and Virtualization: http://for508.com/fj8z6
[2] Inside Windows 7 User Account Control: http://for508.com/mrs1j
[3] Introducing Managed Service Accounts: http://for508.com/z1yui
[4] Update to Improve Credentials Protection and Management: http://for508.com/gakif
[5] Configuring Additional LSA Protection: http://for508.com/a1xiz
[6] Protected Users Security Group: http://for508.com/jare0
[7] Protect Derived Domain Credentials with Windows Defender Credential Guard: http://for508.com/wt28b

Compromising Credentials: Hashes

Hashes

Tokens

Cached Credentials

LSA Secrets

Tickets

NTDS.DIT

The password for each user account in Windows is stored in multiple formats: LM and NT hashes are most well-known. **`TsPkg`**, **`WDigest`**, and **`LiveSSP`** can be decrypted to provide plaintext passwords (prior to Win8.1)

How are they acquired and used? Hashes are available in the **`LSASS`** process and can be extracted with admin privileges. Once dumped, hashes can be cracked or used immediately in a pass-the-hash attack.

Common tools: **`Mimikatz • fgdump • gsecdump • Metasploit • AceHash • PWDumpX • creddump • WCE`**

When discussing credential compromise, account password hashes are an excellent place to start. Windows stores credentials in a variety of different formats. The most commonly known are LM (deprecated) and NT hashes. Local account password hashes are available in the SAM registry hive and domain account hashes are present in memory during interactive sessions. Several standard attacks allow an attacker access to these credentials: memory extraction directly from LSASS, dumping the LSASS process for offline attacks, and extraction of local account hashes from the SAM hive in memory or on disk. Administrator privileges are required for all of these attacks. Once the hashes are dumped, they can be easily cracked with tools like John the Ripper or via rainbow table pre-computation attacks. Alternatively, they can be used for authentication in their original form (see Pass the Hash below). A domain user account must interactively log on to a system for its hash to be present in memory, and the hashes are only available while the user is logged on (unless applications or processes are still using them after logging off).[1]

Cleartext Passwords

Researchers have also found many other types of stored credentials in memory, some of which can be easily extracted and decrypted, resulting in cleartext passwords. Some of these are a result of single sign-on (SSO) like TsPkg and Wdigest, while another, LiveSSP, is derived from the new Windows "Live" cloud accounts that can now be used to log on. Mimikatz was the first tool to bring many of these vulnerabilities to the mainstream, and now tools like Windows Credential Editor (WCE) can be used to extract them as well.[2]

Pass-the-Hash Attacks

Pass the hash allows an attacker to authenticate using a stolen account hash without ever knowing the cleartext password.[3] This is incredibly useful to an attacker who has managed to capture a highly privileged hash and (sadly) defeats even the longest and most complex passwords (because no cracking is necessary). Several tools facilitate this attack, including the Metasploit PsExec module, WCE, and SMBshell. Pass the hash is limited to NTLM authentication and takes advantage of the fact that only the hash is necessary to respond to an NTLM

challenge-response protocol. While most authentication in a Windows enterprise is now using Kerberos, NTLM is still almost always available. Pass-the-hash attacks typically take advantage of the SMB protocol to map file shares and perform PsExec-style remote execution, but they can also be used by many native tools, including WMI.

[1] Protecting Privileged Domain Accounts: Safeguarding Password Hashes: http://for508.com/7jyl6
[2] Mimikatz a Short Journey Inside the Memory of the Windows Security Service (PDF): http://for508.com/j1t3z
[3] Whitepaper: Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques (PDF): http://for508.com/cxhg2

Credential Availability

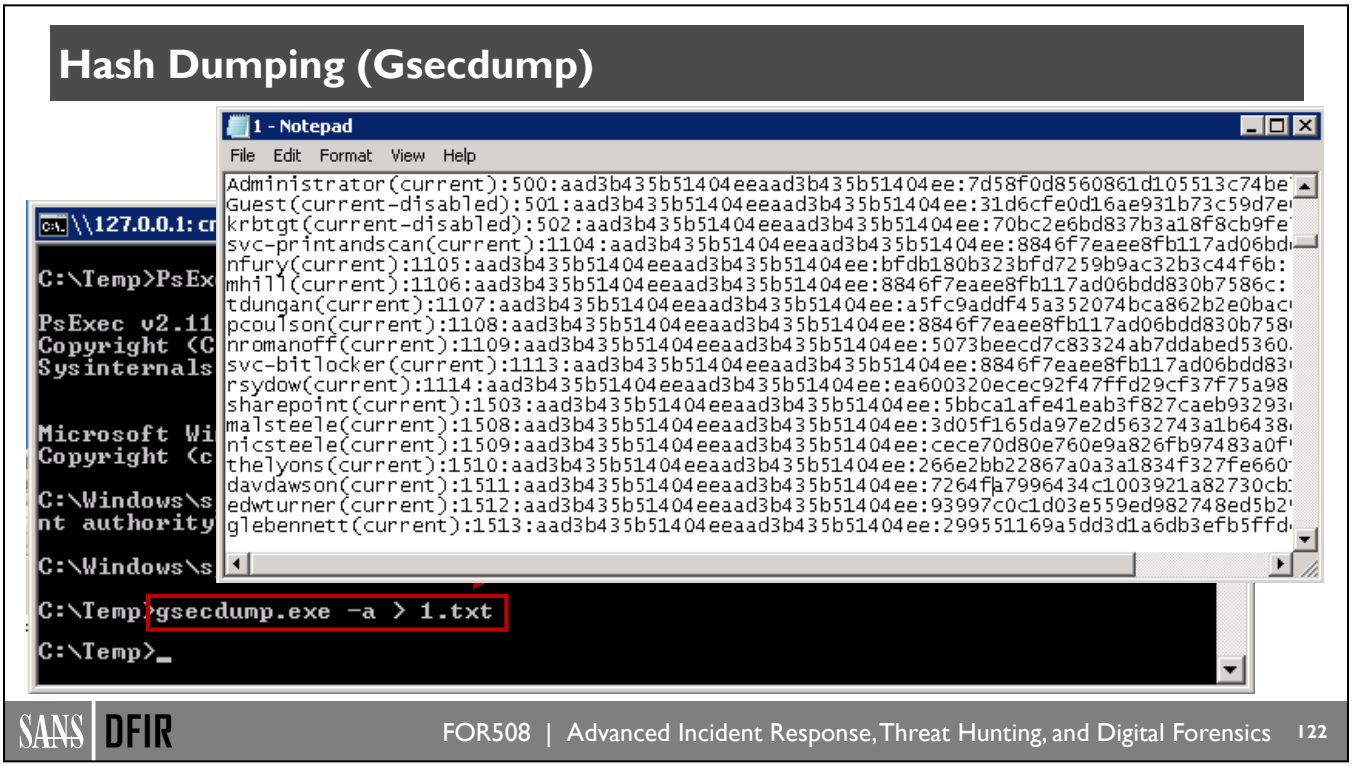
Admin Action	Logon Type	Credentials on Target?	Notes
Console logon	2	Yes*	*Except when Credential Guard is enabled
RunAs	2	Yes*	*Except when Credential Guard is enabled
Remote Desktop	10	Yes*	*Except for enabled Remote Credential Guard
Net Use	3	No	Including /u: parameter
PowerShell Remoting	3	No	Invoke-Command; Enter-PSSession
PsExec alternate creds	3 + 2	Yes	-u <username> -p <password)
PsExec w/o explicit creds	3	No	
Remote Scheduled Task	4	Yes	Password saved as LSA Secret
Run as a Service	5	Yes	(w/user account)—Password saved as LSA Secret
Remote Registry	3	No	

https://technet.microsoft.com/en-us/windows-server-docs/security/securing-privileged-access/securing-privileged-access-reference-material

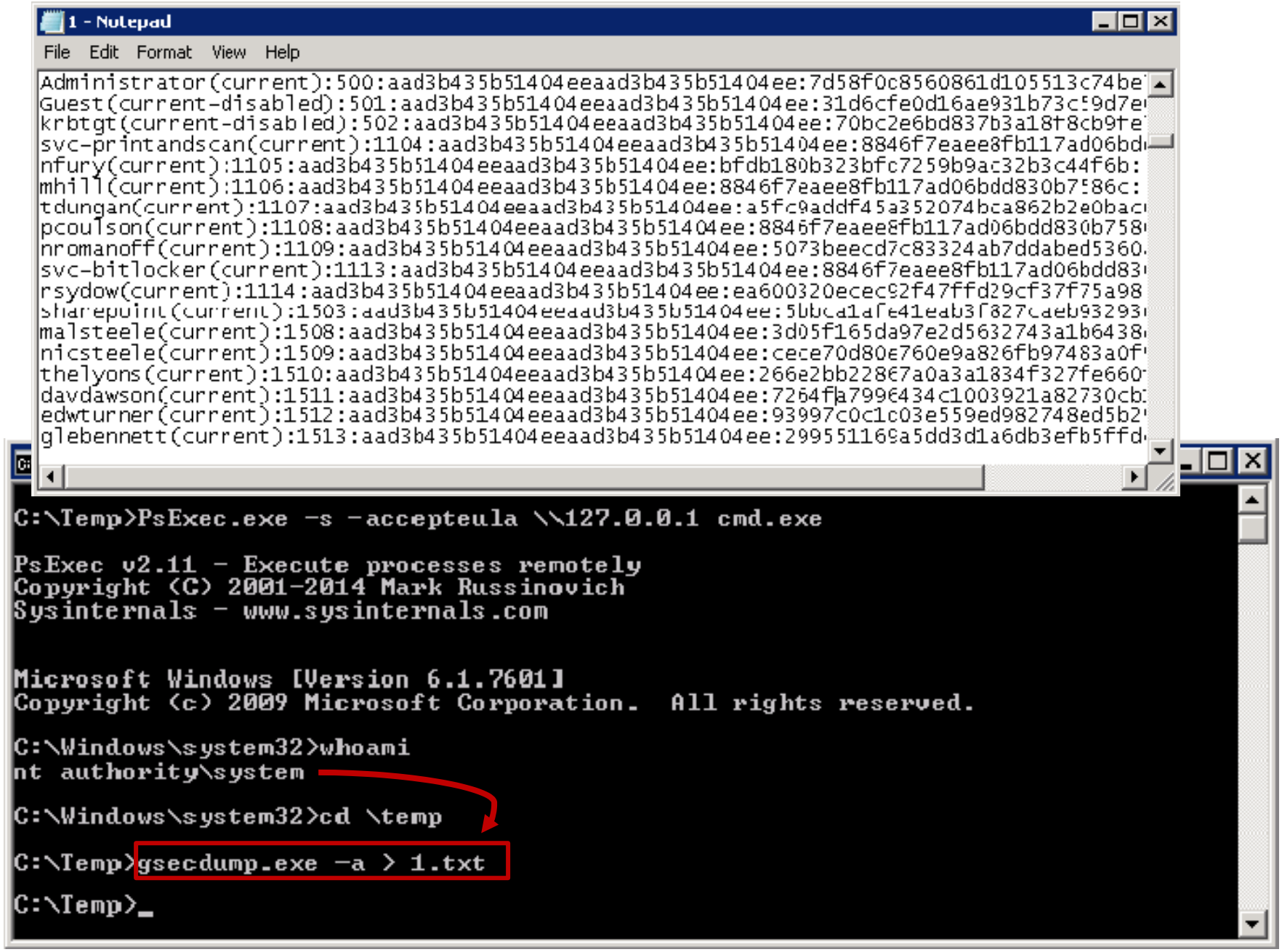
One solution to the Windows credential problem is education. The security community has developed best practices over time for reducing the likelihood of leaving behind high-privileged derived credentials. However, this information has largely not filtered down to the admin community (where it is most important). Very simply, it matters immensely how an administrator authenticates to a Windows system. Interactive logons at the console, via RDP, and using runas, cache hash and ticket credentials on the target system. They should not be used for highly privileged accounts. PowerShell remoting is a much more secure option, as is enabling new Windows 10+ protections like Remote Credential Guard. Even something as simple as adding an explicit username and password (similar to a runas) on the PsExec command line makes the difference as to whether credentials will be available for possible theft. Remote scheduled tasks and services are also very dangerous—we will cover the corresponding LSA Secrets they leave behind later in this section.

This table was inspired by a similar one available on Microsoft TechNet.[1]

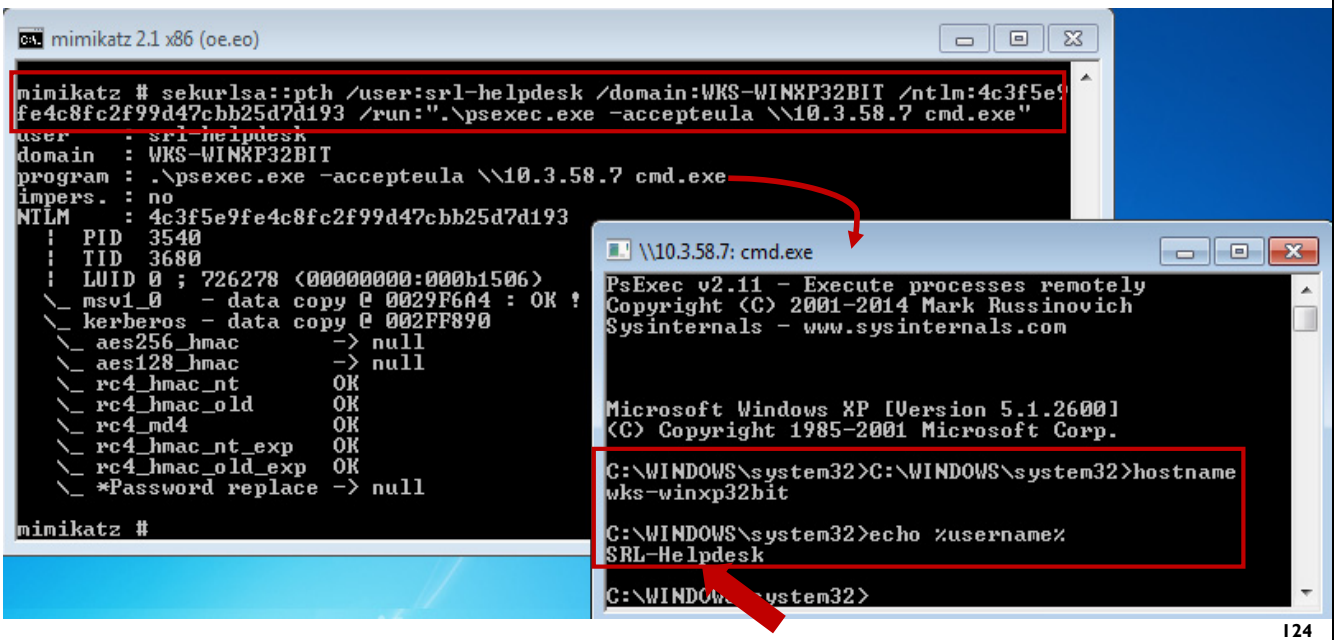
[1] Securing Privileged Access: http://for508.com/eid-r



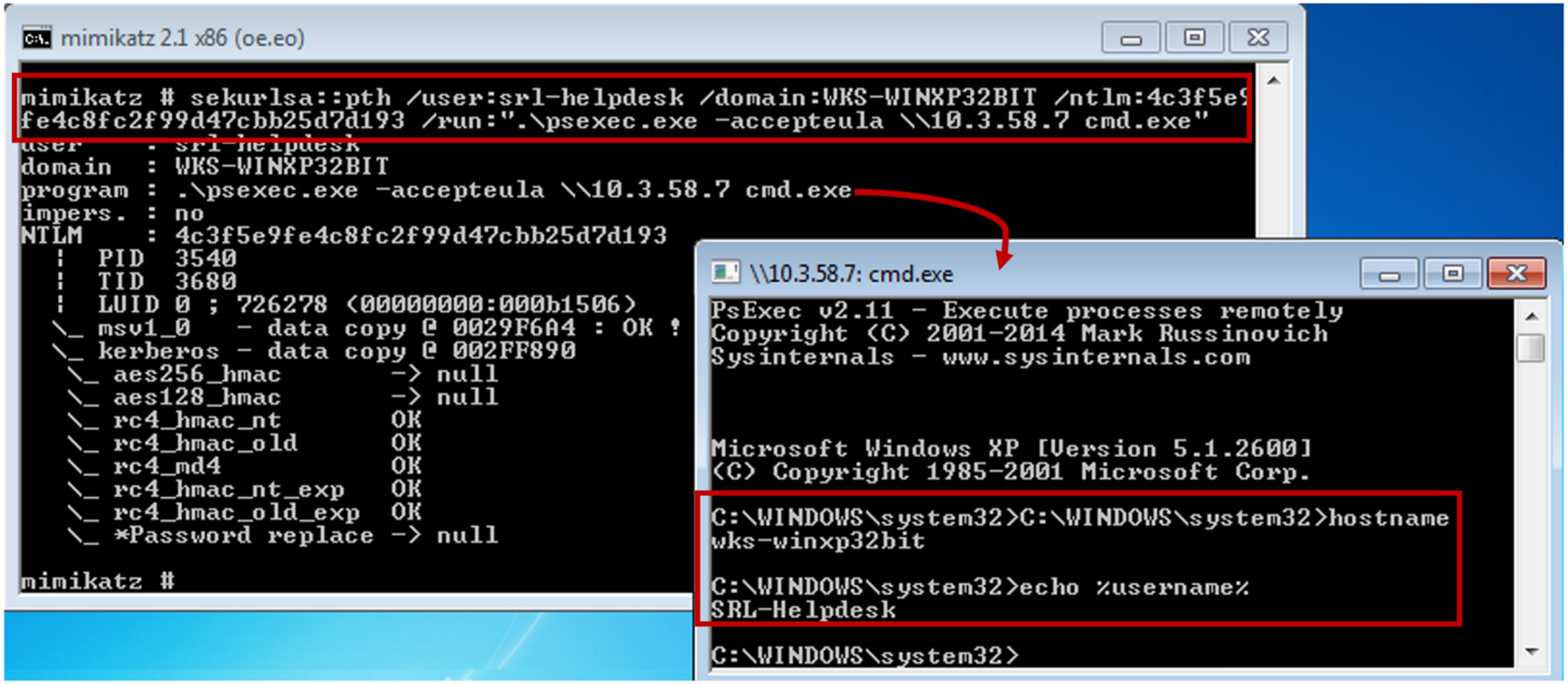
In this example, we see the gsecdump tool executed on a Domain Controller. On a standard system, we would expect to only see hashes available from currently logged on sessions. However, the DC is a special case, as it facilitates many logged on sessions. This is the reason why we often see attackers executing credential dumping tools on DCs, even though it is risky due to increased monitoring and protections. In this example, there were hundreds of hashes available from the gsecdump output.



Pass the Hash (Mimikatz)



Once a Windows NTLM hash is extracted, it can immediately be used to authenticate, with no cracking necessary. In this example, Mimikatz was used with the pth option to authenticate to a remote system (10.3.58.7). The srl-helpdesk account hash successfully authenticated to the remote system, opening an interactive command shell via the PsExec tool. Note that the latest versions of Mimikatz have replaced the "pth" command with functionality that technically performs an attack called "Overpass the Hash", which we will cover later in this section. From the user perspective the results are the same, but if you are ever trying to simulate true pass the hash for testing tools or defenses, Metasploit keeps true to the original implementation.



Defending Credentials: Hashes

- Prevent admin account compromise
- Stop remote interactive sessions with highly privileged accounts
- Proper termination of RDP sessions
  - Win8.1+ → force the use of Restricted Admin?
  - Win10+ → deploy Remote Credential Guard
- Upgrade to Windows 10+
  - Credential Guard
  - TsPkg, WDigest, etc.: SSO creds obsolescence
  - Domain Protected Users Group (PtH mitigation)

The best way to protect hashes is to not interactively log in to systems with highly privileged accounts and stop attackers from gaining local administrative rights that can allow credential stealing. Hashes are only present during interactive logon sessions. Thus, avoiding console logons, RDP, and runas will ensure hashes are not at risk. This is most important on systems like servers, where multiple users may log on, and attackers can easily wait in hiding for a "high-value" account to log in. Perhaps even more common are hashes stolen due to improper termination of RDP sessions. Hashes persist on a system while an interactive session is open. If a session is disconnected, but not closed, the hash can persist on a system while that session is still present. Group Policy can be used to "set a time limit for disconnected sessions", effectively terminating old RDP sessions. Note that if this value is set too low, it could result in lost work due to normal network connection issues.

Microsoft has recently taken major steps to reduce hash vulnerabilities. Windows 8.1+ no longer stores WDigest and TsPkg credentials by default. However, WDigest plaintext credentials can be re-enabled by an attacker by updating **`HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest`** with a "1" in registry value **`"UseLogonCredential"`**. [1] This would be an excellent key to audit for any unexpected changes or additions (the value is not present on Win8.1+ systems by default). Windows 8.1 also introduced several pass-the-hash mitigations, including Restricted Administrator accounts, the Domain Protected User security group, and Protected Processes.[2] Windows 10 introduced Credential Guard and Remote Credential Guard. Restricted Admin accounts and systems with Remote Credential Guard enabled do not make hashes (or tokens) available on the remote system, even during interactive logons like RDP. The Domain Protected Users group cannot authenticate via NTLM, WDigest, or SSP, and it does not cache credentials. Protected Processes will only execute signed code, breaking many current code injection techniques used to dump credentials from LSASS (if enabled as a protected process). Local administrator accounts can be easily restricted from authenticating over the network—including Pass the Hash—unless they happen to be the RID 500 built-in administrator account.[3] Credential Guard debuted with Windows 10, which uses a hypervisor to move domain credentials, tickets, and hashes from LSASS into an isolated process (LSAIso) that can better protect them. All of these mitigations can be defeated, but in short, it is now <u>much</u> more difficult to gather hashes and execute pass-the-hash techniques on Win8.1+ systems.

A final defense is to ensure that local administrator account passwords are unique and not shared across the workstation segment. Microsoft released the Local Administrator Password Solution (LAPS) to centralize security and management of these accounts within Active Directory.[4] Hashes become much less useful when they cannot be used elsewhere.

[1] Microsoft Security Advisory Update to Improve Credentials Protection: https://for508.com/q1v9h
[2] TechNet Credentials Protection and Management: http://for508.com/306oh
[3] Blocking Remote Use of Local Accounts: http://for508.com/zve0b
[4] Local Administrator Password Solution: http://for508.com/mbrx6

Compromising Credentials: Tokens

Hashes

Tokens

Cached Credentials

LSA Secrets

Tickets

NTDS.DIT

Delegate tokens are powerful authentication resources used for SSO. They allow attackers to impersonate a user's security context, including over the network.

How are they acquired and used? The SeImpersonate privilege lets tokens be copied from processes. The new token can then be used to authenticate as the new user. A target user or service must be logged on or have running processes.

Common tools: `Incognito • Metasploit • PowerShell • Mimikatz`

Every Windows logon and process has an associated security token. A token contains security context and privileges for an account. Special tokens, named impersonate and delegate, facilitate access control and single sign-on (SSO), specifically allowing processes to be run under a different account security context. Impersonate tokens allow local security context shifts, and the more powerful delegate tokens facilitate authentication even across network resources. The latter are commonly present during interactive logons, but certain non-interactive Microsoft services also rely upon delegate tokens.[1]

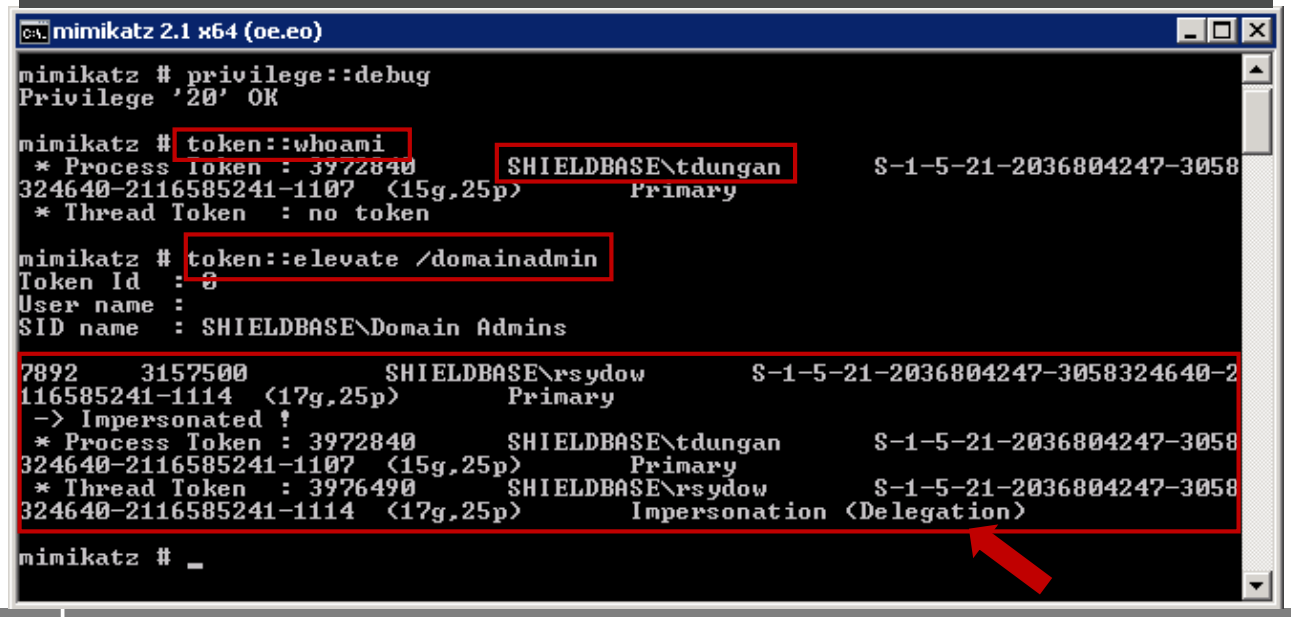
Token Stealing

If a token is present on the system, a user with the SeImpersonate privilege (or with the admin or SYSTEM privileges necessary to add it) can extract the token and reuse it to do a variety of tasks like local privilege escalation, adding users or managing group membership, and mapping remote admin shares or running PsExec (delegate tokens only).[2] Attackers commonly use this technique to elevate privileges from local admin to domain admin. The attack is also particularly useful in situations where hashes cannot be easily extracted from LSASS, such as when it is designated as a Win8.1+ Protected Process.

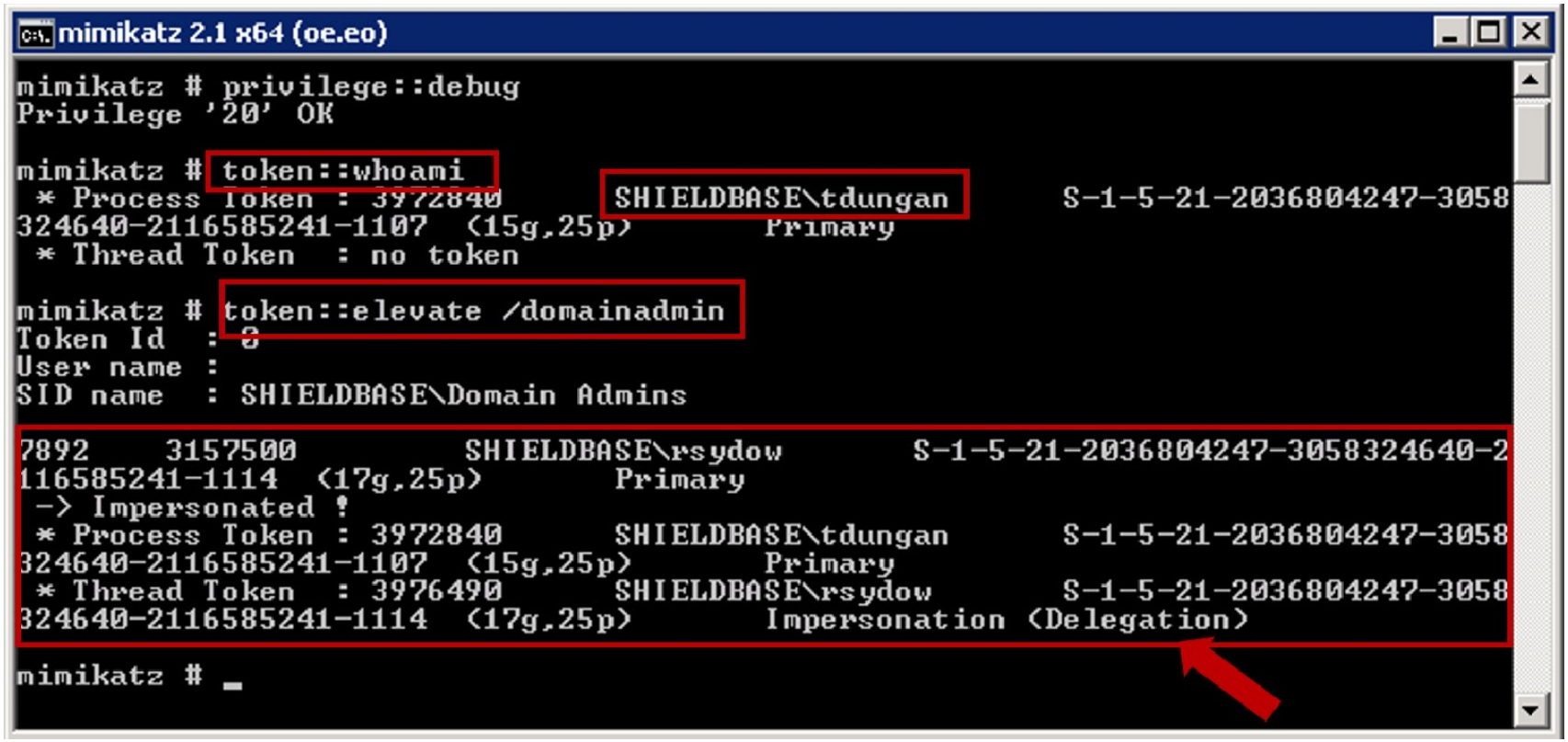
In modern Windows systems (post-Windows 2003SP1), tokens are present only when an account is logged in. Servers are a particularly lucrative place to find them since they may have simultaneous users. Another commonly exploited vector is on systems where an administrator has connected via RDP but did not perform a proper session termination (i.e., they closed the RDP client without logging out).

[1] Protecting Privileged Domain Accounts: Safeguarding Access Tokens: http://for508.com/zmyn0
[2] Security Implications of Windows Access Tokens (PDF): http://for508.com/6zb1t

Token Stealing (Mimikatz)



A classic token stealing attack is shown here. The attacker was authenticated using a local admin account named "tdungan". The Mimikatz tool was run with the built-in command "token::elevate /domainadmin". This command identifies any domain administrator tokens present on the system, retrieves them, and loads them into process memory so the higher-level privileges can be used.



Defending Credentials: Tokens

- Prevent admin account compromise
- Stop remote interactive sessions with highly privileged accounts
- Proper termination of RDP sessions
  - Win8.1+ → force the use of Restricted Admin Mode?
  - Win10+ → deploy Remote Credential Guard
- Account designation of "Account is Sensitive and Cannot be Delegated" in Active Directory
- Domain Protected Users security group accounts do not create delegate tokens

Similar to password hashes, the best way to protect sensitive tokens is to not interactively log in to systems with highly privileged accounts and stop attackers from gaining local administrative rights that can allow token stealing. Coveted "delegate" tokens are only present during interactive logon sessions.[1] Thus, avoiding console logons, RDP, and runas largely removes the ability to steal delegate tokens (note that Mike Pilkington discusses a few edge cases where network logons can also utilize delegate tokens[2]). Delegate tokens are most dangerous on systems like servers, where attackers know multiple users may log on and can easily wait in hiding for a "high-value" account to log in. Perhaps even more common are tokens stolen due to improper termination of RDP sessions. If a session is disconnected but not closed, the token can persist on a system while that session is still present. Group Policy can be used to "set a time limit for disconnected sessions", effectively terminating old RDP sessions. Note that if this value is set too low, it could result in lost work due to normal network connection issues.

Luckily, there are more rigorous ways to lock down delegate tokens rather than relying on good user behavior. In Windows 8.1+, forcing an account to use "Restricted Admin" prevents hashes and tokens from being available on the remote system during interactive logons like RDP.[3] High-value accounts can also be designated in Active Directory as not able to be delegated.[4] Make sure to test this, as it will break some capabilities of these accounts (some infrastructure relies upon token delegation) and may not be feasible for all highly privileged accounts. This exercise might help with the development of a tiered account approach, helping isolate dangerous accounts using least privilege methodology. Remote Credential Guard, introduced in Windows 10/Server 2016, was designed to fix some of the issues with Restricted Admin, including the delegation issue for third-party resources and protection of non-admin accounts.

Perhaps the most elegant solution now exists in Windows 8.1+ systems. Members of the Protected Users security group do not create delegate tokens, even during interactive sessions. Again, these accounts will not be able to be used for every administrative function, but this group is a great place for generic domain-admin level accounts.[5]

[1] Monitoring for Delegation Token Theft: http://for508.com/4nv9m
[2] Protecting Privileged Domain Accounts: Safeguarding Access Tokens: http://for508.com/6c0n3
[3] TechNet Credentials Protection and Management: http://for508.com/k69i3
[4] Analyzing “Account is sensitive and cannot be delegated”: http://for508.com/khltc
[5] TechNet Protected Users Security Group: http://for508.com/hm6po

Compromising Credentials: Cached Credentials

Hashes

Tokens

Cached Credentials

LSA Secrets

Tickets

NTDS.DIT

Stored domain credentials to allow logons when domain controller access is unavailable. Most systems cache the last 10 logon hashes by default.

How are they acquired and used? Cached credentials must be cracked. Hashes are salted and case-sensitive, making decryption very slow. These hashes cannot be used for pass-the-hash attacks.

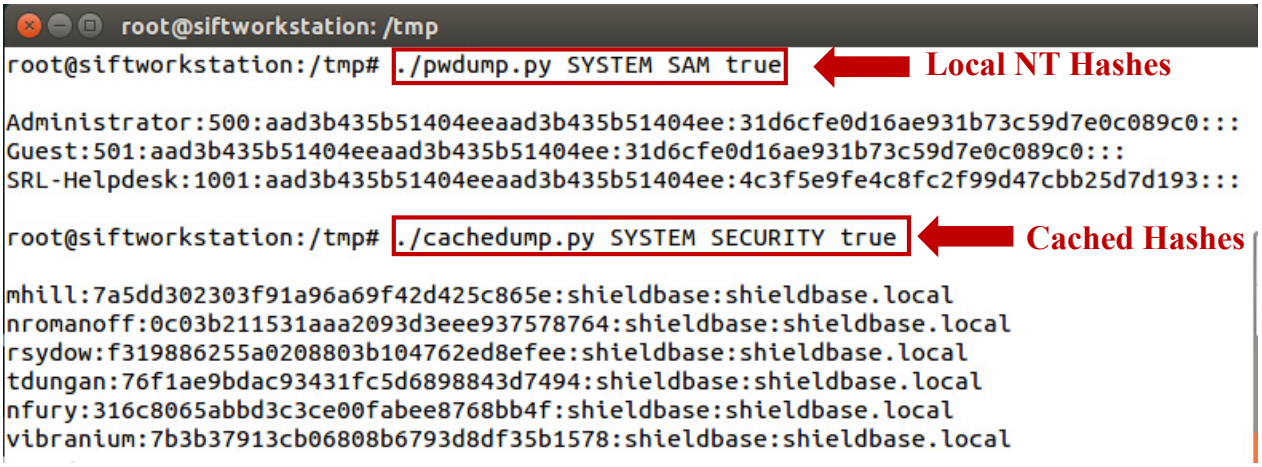
Common tools: `cachedump` • `Metasploit` • `PWDumpX` • `creddump` • `AceHash`

In a domain environment, the domain controller (DC) is responsible for authenticating accounts. But what happens if the system is offline or cannot communicate with the DC? To prevent a situation where a user cannot log on to the system, Windows caches the last ten logon hashes by default (amazingly, this number was increased to 25 for Server 2008).[1] Imagine how many accounts log on to the typical workstation. Probably less than ten! Credentials can persist in this cache for very long periods. Attackers hope that at some time in the past, a domain administrator (or similarly highly privileged account) logged on to the system interactively and had their credentials stored indefinitely.

Cached domain credentials are stored in the Security registry hive in the SECURITY\Cache key. Administrator (or System) privileges are required to access the saved hashes, which are in mscash2 format for modern Windows operating systems. Since data is kept in the registry, these hashes persist indefinitely, even after a reboot. The tool creddump can extract hashes offline, from exported registry hives. Mscash2 hashes are encrypted and thus cannot be used in pass-the-hash attacks. They are also secured better than NT hashes, including a hash salt of the username to defeat pre-computation attacks. Tools like John the Ripper and hashcat can brute force crack these hashes, but with a good password policy in place, decryption can be very slow. Note that some of the older hack tools in this category, notably cachedump and PWDumpX, appear to have problems extracting cached credentials on systems post-WinXP.

[1] Cached and Stored Credentials Technical Overview: http://for508.com/bfwkn

Offline Cached Credentials Extraction (Creddump)



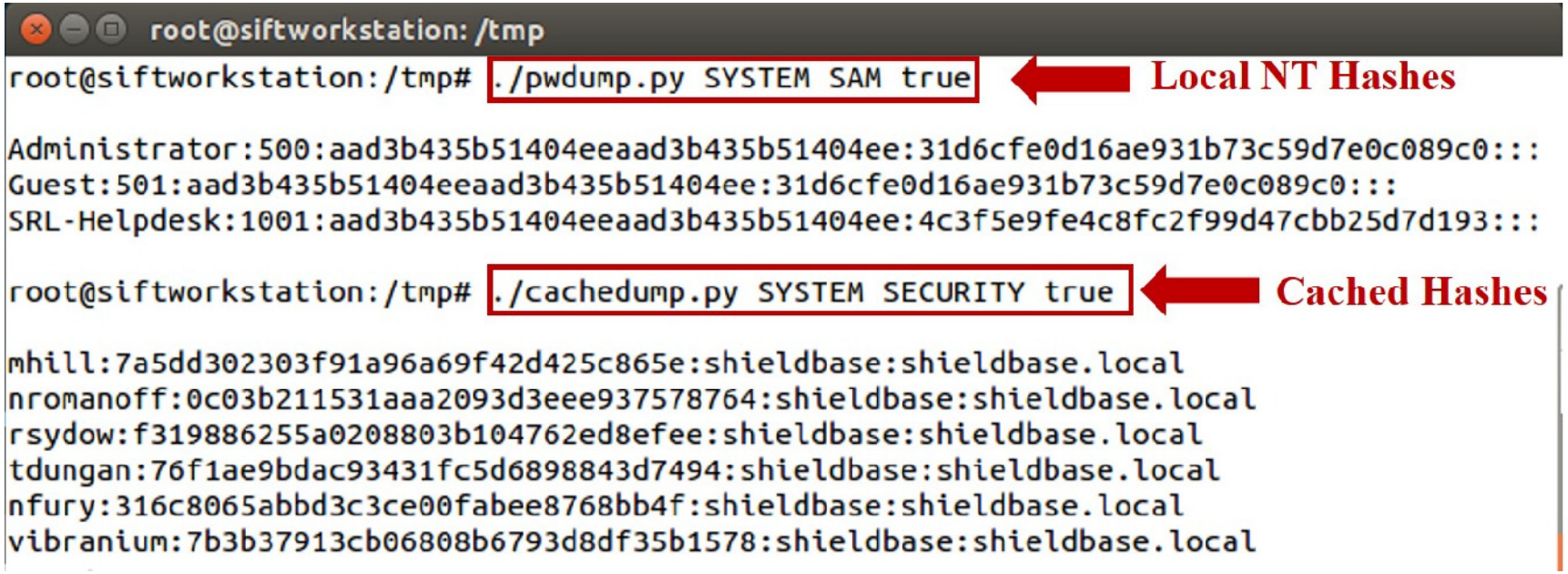
The creddump utilities can extract hashes, cached credentials, and LSA Secrets from offline registry hives: github.com/Neohapsis/creddump7

In this example, we see two tools from the creddump suite being used to extract hashes and cached credentials from offline registry hives. Offline credential extraction may be particularly interesting to attackers who are wary of running common credential dumping binaries on a system. Instead of taking the risk of a security tool identifying the binary, injection attempt, new service creation, or other artifacts left behind by malicious tools, the attacker can simply copy the registry hives and extract credential data at a later time.

There are many offline credential extraction tools, including Mimikatz, which requires a process dump of lsass.exe. Creddump is one of the more reliable tools available and is written in Python. Creddump was originally created as a proof of concept by Brendan Dolan Gavitt and was subsequently updated and patched by Ronnie Flathers.[1]

Of particular interest in this slide is the difference between dumping local hashes and cached credentials. Pwdump.py was used to extract hashes from the SAM hive. These are local accounts that belong to this system. Cachedump.py was subsequently used to extract cached credentials. These are potentially much more interesting to an attacker since they are domain credentials and not subject to prevalent local account restrictions. There are also several to choose from! Cached credentials are salted with the account username and thus must be cracked via brute force. This tool nicely outputs them in the proper format to feed to the John the Ripper password cracking tool (the format is named mscash2). While cached credentials certainly take longer for an attacker to leverage, they have good potential to provide the highly privileged domain accounts that attackers seek.

[1] Creddump: http://for508.com/l-em1



Defending Credentials: Cached Credentials

- Prevent admin account compromise
- Limit number of cached logon accounts
  - SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon (*cachedlogonscount* value)
  - A *cachedlogonscount* of zero or one is not always the right answer
- Enforce password length and complexity rules
  - Brute force cracking is required for this attack
- Domain Protected Users security group accounts do not cache credentials

Like many mitigations for credential access, preventing an attacker from achieving administrator permissions removes their ability to retrieve cached credentials from the Security hive. Since this has thus far proven nearly impossible in practice, other mitigations are worth exploring.

A common mitigation for this attack is to reduce the default number of cached logons in the registry key SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\. By default, Windows workstation operating systems cache the last 10 logons and some servers cache up to 25.[1] US Department of Defense Security Technical Implementation Guides (STIGS) commonly recommend this value be four or less.[2] Some organizations set this value to zero or one (NSA recommended a value of zero for non-mobile XP systems). But be careful when endorsing a very low number of cached credentials. The cachedlogonscount value is not only for user accounts. Service accounts and computer account logons also get cached. Smart card usage can count as two cached logons. If you set the value to one or zero, it may not cache the standard user account for offline use.

Microsoft began to address this threat starting with Windows 8.1. The introduction of the Protected Users security group provides additional non-configurable credential protections. One of these protections is that credentials are not cached for members of this group. The group is designed for high-value accounts and is a significant step toward mitigating credential theft.

[1] Cached logons and CachedLogonsCount: http://for508.com/q09hg
[2] Domain Controller Security Technical Implementation Guide: http://for508.com/lwutc

Compromising Credentials: LSA Secrets

Hashes

Tokens

Cached Credentials

LSA Secrets

Tickets

NTDS.DIT

Credentials stored in the registry to allow services or tasks to be run with user privileges. In addition to **service accounts**, may also hold application passwords like VPN or auto-logon credentials.

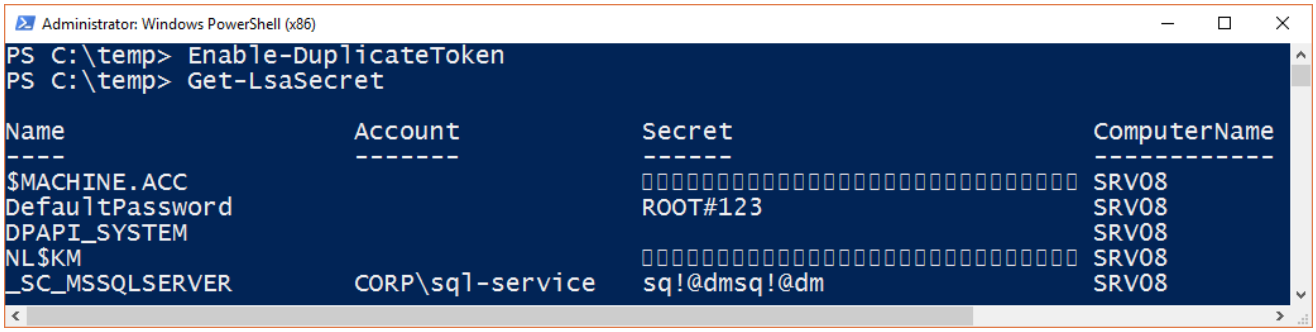
How are they acquired and used? Administrator privileges allow access to encrypted registry data and the keys necessary to decrypt. Passwords are plaintext.

Common tools: `Cain` • `Metasploit` • `Mimikatz` • `gsecdump` • `AceHash` • `creddump` • `PowerShell`

Windows services are designed to run without user interaction and thus when they are executed using domain accounts (as opposed to local built-in accounts like Local System, Network Service, and Local Service), the password for that account must be stored somewhere. That somewhere is in the LSA Secrets key within the Windows registry. Over the years, various applications (including Windows) have found it useful to store a wide variety of different credentials in this "protected" area, including RAS and VPN passwords, default logon credentials, credentials used to authenticate scheduled tasks, and even IIS application passwords. From an attacker perspective, service accounts are by far the biggest draw, as they are very common in the Windows enterprise, are often highly privileged, and rarely change.

LSA Secrets are stored in encrypted form in the Security hive registry key SECURITY/Policy/Secrets. Each secret has its own key, and a parent key within SECURITY/Policy can decode the "secrets". In addition to the Security hive, dumping tools require access to the System hive to access a final decryption key. Administrator privileges allow access to all of the registry areas necessary to dump the LSA Secrets. Once decrypted, all passwords are in cleartext, making them particularly valuable for use with applications like RDP.

Decrypting LSA Secrets (Nishang)



Get-LsaSecret.ps1 from the Nishang PowerShell pentest framework used to dump (and decrypt) LSA Secrets

https://github.com/samratashok/nishang

LSA Secrets are stored in encrypted form within the Registry. However, the keys are available with admin rights, and the format and locations of the data are well documented. There are many tools that have been developed over the years to exploit LSA Secrets. While any pen tester will tell you that the data stored can often be very boring, once in a while an attacker hits the lottery and finds a domain admin service password nicely in plaintext.

This example shows an interesting update to the wealth of LSA Secret dumping tools—implementation coded as a PowerShell script. The Nishang framework is designed for offensive security operations and contains many different scripts for attacking systems. Here we see the Get-LsaSecret script being used to dump the registry data. Note that this particular script requires an administrator 32-bit PowerShell console (which is available on every modern Windows OS). The script run prior to this one, Enable-DuplicateToken, is also part of Nishang and is required to gain permissions necessary to access the Security registry hive. It works by setting the current process thread token to the same token currently in use by the LSASS process (very clever).

The output on this slide represents exactly what an adversary hopes to find—a domain service account (sql-service) and password (sq!@dmsq!@dm). If this account ends up having domain admin privileges, the attacker is well on their way to owning the enterprise. Note that most of the credential attacks discussed in this section show that password complexity is rarely enough to prevent compromise. This example demonstrates this, as the sql-service password is 12 characters and relatively complex. The “DefaultPassword” value is used when auto-login is enabled on the system. The value “ROOT#123” is a common default on Windows servers for this feature and likely means that auto-login is not in use for this system (it is rarely in use in an enterprise). The non-printable characters shown for $MACHINE.ACC and NL$KM are actually strings of hex values stored for those “secrets”. They are not passwords.

Defending Credentials: LSA Secrets

- Prevent admin account compromise
- Do not employ services or schedule tasks requiring privileged accounts on low-trust systems
- Reduce number of services that require domain accounts to execute
  - Heavily audit any accounts that must be used
- (Group) Managed Service Accounts

The best option for mitigating LSA Secrets attacks is ensuring low-trust systems do not have services or scheduled tasks present that require highly privileged, non-built-in accounts. When this situation occurs, those privileged accounts must be stored in the LSA Secrets registry key, making them available to any attacker with admin rights on the system. The use of these service accounts and tasks can creep into an organization and are pernicious to the environment once a compromise occurs.

Auditors should report on the existence of services or tasks that rely upon domain accounts, and infrastructure changes often must be made in order to reduce the reliance on these very dangerous implementations. If a vendor requires the use of a highly privileged service account, you should force them to implement a new solution or look for other options.

Since it can be very difficult to totally remove the reliance on LSA Secrets, any accounts that are likely to be present in this key should follow the least privilege rule and be heavily audited.

Managed Service Accounts were released with Server 2008R2. This feature specifically helps with service accounts being run with domain user rights. It provides both frequent password changes (30 days by default) and long and complex passwords for these accounts. While an excellent idea, the initial implementation was lacking. The latest iteration, Group Managed Service Accounts, is much more flexible and admin-friendly.[1]

[1] Avoid password management with Group Managed Service Accounts: http://for508.com/80l56

Compromising Credentials: Tickets

Hashes

Tokens

Cached Credentials

LSA Secrets

Tickets

NTDS.DIT

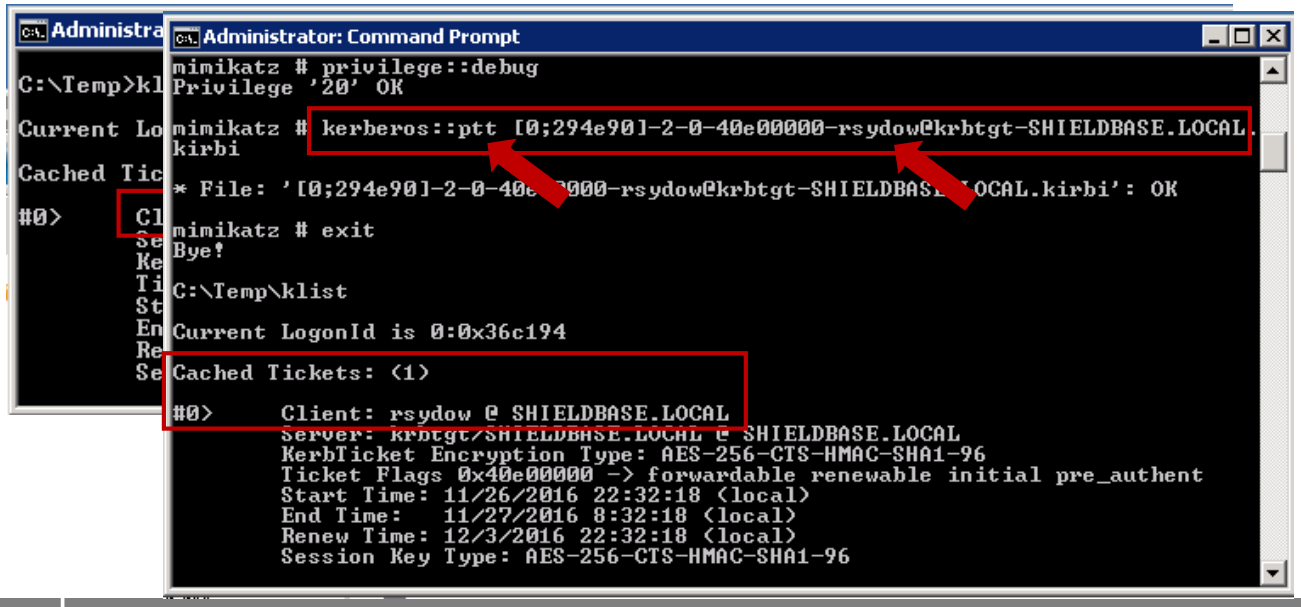
Kerberos issues tickets to authenticated users that can be reused without additional authentication. Tickets are cached in memory and are valid for 10 hours.

How are they acquired and used? Tickets can be stolen from memory and used to authenticate elsewhere (Pass the Ticket). Further, access to the DC allows tickets to be created for any user with no expiration (Golden Ticket). Service account tickets can be requested and forged, including offline cracking of service account hashes (Kerberoasting).

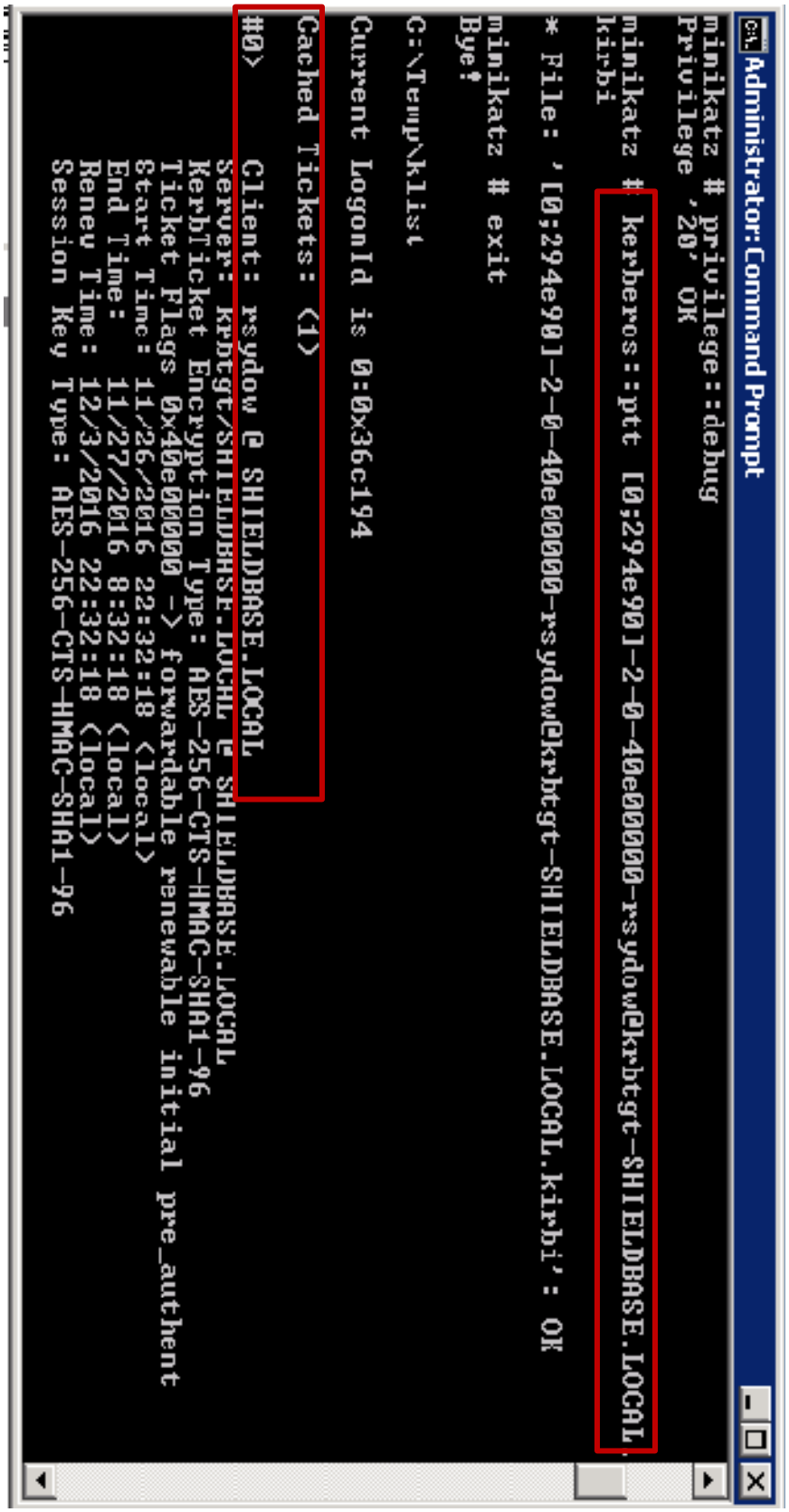
Common tools: `Mimikatz` • `WCE` • `kerberoast`

The Windows enterprise relies heavily on the Kerberos authentication protocol. This protocol uses Ticket Granting Tickets (TGTs) to prove successful authentication of user accounts and service tickets to authenticate for particular services. Since nearly everything in Windows is tied to an account, the authentication burden can be massive. To reduce the number of authentication requests that bombard a domain controller (DC), issued tickets are valid for 10 hours by default. This means that anyone in possession of that ticket is already pre-authenticated during that time period. Attackers can use tickets to impersonate privileged users, evade the authentication process, and reduce logging of their efforts. Attack details will be covered in an upcoming slide.

Pass the Ticket (Mimikatz)



In this example, the attacker previously dumped available tickets using the Mimikatz option "sekurlsa::tickets /export". This writes each ticket out to a file, which can then be subsequently moved and imported to wherever it will be useful. The Mimikatz "kerberos::ptt" option is seen here importing a ticket for a domain admin user named "rsydow". After importation, the built-in Windows command "klist" is used to show that the desired ticket is cached on the system and available to authenticate throughout the environment.



Kerberos Attacks

Pass the Ticket	Steal ticket from memory and pass or import on other systems
Overpass the Hash	Use NT hash to request a service ticket for the same account
Kerberoasting	Request service ticket for highly privileged service and crack NT hash
Golden Ticket	Kerberos TGT for any account with no expiration. Survives full password reset
Silver Ticket	All-access pass for a single service or computer
Skeleton Key	Patch LSASS on domain controller to add backdoor password that works for any domain account
DCSync	Use fake Domain Controller replication to retrieve hashes (and hash history) for any account without login to the DC

Pass the Ticket
Ticket attacks are not new, but recent improvements in tools have made them easier to steal and employ. Tickets are cached in memory during the time they are valid, and a tool like Mimikatz allows an attacker with administrator privileges to dump all of the tickets present on the system to individual files. They can then pass them to other systems to authenticate as that account (TGT) or service. Tickets can also just be imported into other systems, allowing a user "transplant" without requiring any knowledge of the user hash or password.[1]

Overpass the Hash (Pass the Key)
If an attacker can dump an account hash from the system, this hash can be used to request service tickets, providing access to other system resources (like file shares). While similar to pass the hash, the attack employs Kerberos for authentication (instead of NTLM), meaning it can work even when pass-the-hash mitigations are in place (like limiting NTLM authentication over the network).

Kerberoasting
Interestingly, any domain user can request a ticket from the Domain Controller for any domain service. The ticket returned for such a request has a non-salted password hash for the account that runs that service. Attackers can seek tickets for services being run under the context of a domain user (or even better domain admin) accounts and can derive the plaintext account password. The encrypted hash can be cracked offline, all without any failed logon attempts or account lockouts! Since service account passwords often have high privileges and are rarely changed, these passwords can be very valuable.

Golden Ticket
A "Golden Ticket" is a method of creating Kerberos TGT authentication tickets that do not expire (Mimikatz sets the default expiration to 10 years, but this can be increased). Golden Tickets can be created for any account, and are commonly used to create a persistent domain administrator TGT that will work even after a full

password reset of the enterprise.[2] This attack gives an adversary an unprecedented level of persistence. If an attacker loses access, all that is required is user-level access on any system and the previously generated Golden Ticket can be re-introduced to elevate to domain administrator again using the Pass the Ticket attack. There are few Windows logging events that can track the use of a Golden Ticket.

To create a Golden Ticket, the attacker must first achieve administrator credentials on the domain controller. They can then extract the hash for the krbtgt account, the account used to create tickets. The hash can be extracted from the DC memory, or via the theft of the NTDS.DIT active directory database. After that is accomplished, the creation of the ticket can be accomplished offline at the attacker's leisure.

Silver Ticket

By dumping the computer account hash from memory, an attacker can create a "Silver Ticket" for that system. Think of a Silver Ticket as an all-access pass for a single service or computer. This attack forges a service ticket using a dumped computer account hash, and can (usually) impersonate any user for that system, including domain admin. They also become excellent backdoors, since authentication occurs without requiring communication with the Domain Controller (minimizing log evidence). Computer account passwords are supposed to be changed every 30 days but don't always get changed. Attackers have also been known to turn off password updates on systems to ensure Silver Tickets work indefinitely.

Skeleton Key

Once an attacker has access to the Domain Controller, a tool like Mimikatz (or other malware found in the wild) can patch LSASS to enable a backdoor password for any valid domain user. A user's actual password can still be used to authenticate, but accounts will also authenticate with the "skeleton key" backdoor password, even after a user changes their password. This attack provides an easy and persistent way to abuse accounts throughout the active directory environment.

DCSync

Domain controllers (DC) frequently communicate with one another, synchronizing and updating data using the protocol MS-DRSR (Directory Replication Service Remote Protocol). Security researchers Benjamin Delpy and Vincent Le Toux discovered multiple ways to abuse this protocol, including the ability to impersonate a DC and send a "sync" request to a legitimate DC for the password hash (and hash history) of an account of their choosing. The requested account can even be the KRBTGT ticket necessary to create Golden Tickets! This attack requires high level privileges -- directory replication rights that often come by default as a member of the Administrators, Domain Administrator, or Domain Controllers Group. A significant advantage of DCSync is it works remotely without requiring an interactive logon with the DC. Another attack was subsequently released, DCShadow, using the same protocol to inject data into the Active Directory infrastructure to maintain persistence or setup more advanced attacks.

[1] Protecting Windows Networks – Kerberos Attacks: http://for508.com/l-zh8
[2] Protection from Kerberos Golden Ticket (PDF): http://for508.com/i3-4q

Defending Credentials: Tickets

- Credential Guard (Win10+)
  - Domain Protected Users Group (Win8+): Some attacks
- Remote Credential Guard (Win10+)
  - Restricted Admin (Win8+)
- Long and complex passwords on service accounts
  - Change service account passwords regularly
  - Group Managed Service Accounts are a great mitigation
- Audit service accounts for unusual activity
- Limit and protect Domain Admin
  - Change KRBTGT password regularly (yearly)

Ticket-based attacks are one of the most difficult credential issues to detect and mitigate because most ticket attacks use valid "features" of Kerberos. Tickets can remain valid for some time, even after a password change occurs, and since tickets are generated after multifactor authentication is employed, that control is effectively bypassed as well. It wasn't until Windows 10 and Server 2016 that any real mitigations were available.

Credential Guard in Windows 10+ uses virtual machine isolation to prevent access to a user's hashes and tickets. This prevents even an admin user from dumping tickets to use for attacks like pass the ticket and/or dumping hashes to use for overpassing the hash (AKA Pass the Key) attacks. It is solid protection and an excellent reason to migrate the enterprise to Windows 10+. The Domain Protected Users security group that debuted in Windows 8.1 can prevent attacks like overpass the hash since it eliminates the ability to use the legacy RC4 encryption scheme. Tickets also have a shorter lifespan (4 hours) by default, but pass the ticket attacks cannot be completely prevented since the user's ticket must still be present in memory.

Remote Credential Guard and Restricted Admin are helpful in mitigating the threat since user credentials are not passed to remote systems, even during interactive sessions like RDP. Thus, tickets for high-value accounts cannot be stolen from remote systems when these options are enabled. This helps prevent pass the ticket attacks.

Even with Credential Guard and Remote Credential Guard, there is still one gaping hole in Kerberos security. While user account tickets are very well protected with these new features, service tickets can still be abused. One demonstrated attack against Remote Credential Guard is to use the current user's token to request arbitrary service account tickets from the Domain Controller, which can then be used to authenticate over the network.[1] A similar attack uses the fact that any domain user can request a ticket from the Domain Controller for any domain service. The ticket has a password hash for the account that runs that service and attackers seek tickets that are being run under the context of domain user (or even better, domain admin) accounts. The hash can then

be cracked offline (Kerberoasting).[2] Since Kerberoasting is an offline cracking attack, it does not result in failed logins. The best defense against these types of attacks is to change service account passwords regularly and use long and complex passwords. The former reduces the lifespan of attacker-collected tickets, and the latter effectively prevents Kerberoasting. A feature first released with Server 2008R2, Managed Service Accounts, is the ideal option since it automatically changes service account passwords every 30 days and uses very large and complex passwords. However, none of these actions can completely prevent service ticket abuse. It is still critical that defenders actively monitor service accounts with high privileges for anomalous activity.

The Golden Ticket attack is particularly difficult to identify and mitigate. One way to detect it is to look for invalid accounts being used in the enterprise (though a Golden Ticket can also be created for a valid account). To defeat a Golden Ticket, the krbtgt account password used by the DC to create tickets must be changed (twice).[3] Some organizations are preemptively changing this password on a regular basis to kill any Golden Tickets currently in use.

[1] Exploring the limitations of Remote Credential Guard: http://for508.com/90fi6
[2] GitHub kerberoast: http://for508.com/5acrs
[3] Kerberos in the Crosshairs: Golden Tickets, Silver Tickets, MITM, and More: http://for508.com/2d0ho

Kerberos Attack Mitigations

Attack Type	Description	Mitigation
Pass the Ticket	Steal ticket from memory and pass or import on other systems	Credential Guard; Remote Credential Guard
Overpass the Hash	Use NT hash to request a service ticket for the same account	Credential Guard; Protected Users Group; disable RC4 authentication
Kerberoasting	Request service ticket for highly privileged service and crack NT hash	Long and complex service account passwords; Managed Service Accounts
Golden Ticket	Kerberos TGT for any account with no expiration. Survives full password reset	Protect domain admin accounts; change KRBTGT password regularly
Silver Ticket	All-access pass for a single service or computer	Regular computer account password updates
Skeleton Key	Patch LSASS on domain controller to add backdoor password to any account	Protect domain admin accounts; smart card usage for privileged accounts
DCSync	Use false DC replication to obtain hashes	Protect domain admin; audit/limit accounts with replication rights

The following actions are common mitigation for Kerberos and ticket attacks:

Pass the Ticket: Credential Guard, Remote Credential Guard

Overpass the Hash (Pass the Key): Credential Guard; Protected Users Group; disable RC4 authentication

Kerberoasting: Long and complex service account passwords; Managed Service Accounts

Golden Ticket: Protect domain admin accounts; change KRBTGT password regularly

Silver Ticket: Regular computer account password updates

Skeleton Key: Protect domain admin accounts; smart card usage for privileged accounts

DCSync: Protect domain admin; audit/limit accounts with replication rights

Compromising Credentials: NTDS.DIT

Hashes

Tokens

Cached Credentials

LSA Secrets

Tickets

NTDS.DIT

Active Directory Domain Services (AD DS) database holds all user and computer account hashes (LM/NT) in the domain. Encrypted, but algorithm is well-known and easy to defeat.

How is it acquired and used? Located by default in the `\Windows\NTDS` folder on the domain controller. The file is locked, so admin access is required to load a driver to access raw disk or use the Volume Shadow Copy Service.

Common tools: `ntdsutil` • `VSSAdmin` • `NTDSXtract` • `MetaSploit` • `PowerShell` • `secretsdump.py`

The domain controller is usually one of the first places attackers traverse to after achieving domain administrator credentials. And a primary objective is the NTDS.DIT file. This database has the "keys to the kingdom" and can provide access to nearly every resource in the domain, including those special accounts protecting resources even domain administrators can't access. Account hashes are in NT format (and possibly LM for older systems) and hashes for each account's password history are also present (excellent for taking advantage of password reuse elsewhere).

The NTDS.DIT database is in the well-known Extensible Storage Engine (ESE) format and is located by default in the %SystemRoot%\NTDS folder. While rare, the location of this file can be changed from the default with the registry key HKLM\SYSTEM\CurrentControlSet\Services\NTDS\Parameters recording the new location. It is a locked and protected file, so it cannot be copied via ordinary means. The most common ways to extract the file include using a built-in tool like ntdsutil, loading a driver or tool that gives raw access to the disk (evading Windows API protections), or using the built-in Volume Shadow Copy service. Whatever the technique, admin privileges are required on the domain controller, which is at least a small hurdle.

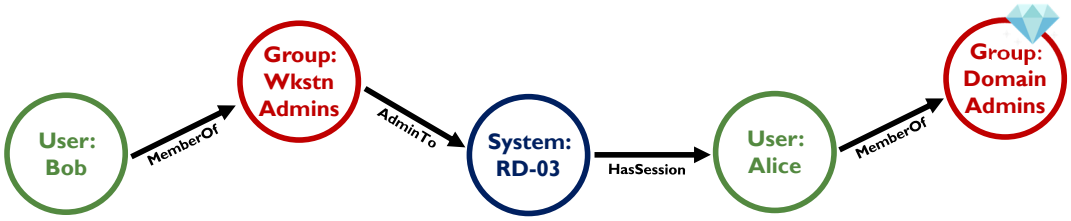
The Volume Shadow Copy extraction technique requires no extra files and is currently the most popular choice in the wild. If there are no Volume Shadow Copies on the system, an attacker can simply create one! The SAM and SYSTEM registry hives must also be collected from the domain controller to facilitate decrypting the data within the database. Once the files are collected, offline extraction can be accomplished using a variety of open-source tools. One of the most famous is the Impacket secretsdump.py script.[1] The result is every NT hash in the domain, which can then be cracked or used immediately via pass-the-hash attacks.

It goes without saying that this attack is by far the most dangerous and devastating that can be accomplished in the enterprise. Retrieving data from the NTDS.DIT requires administrative access to a domain controller (DC). In most cases, if an attacker has achieved this level of access, they are well on their way to owning the entire enterprise. It also opens up attacks like Golden Tickets. Thus, the most relevant mitigation is to detect and stop adversaries before they gain access to the DC.

[1] GitHub impacket/secretsdump.py: https://for508.com/p1xl-

Finding a Path to Domain Admin: Bloodhound

- Bloodhound is an Active Directory relationship graphing tool
  - Nodes: Users, Computers, Groups, Domains, OUs, GPOs
  - Edges: MemberOf, HasSession, AdminTo, TrustedBy, ...
  - Paths: A list of nodes connected by edges (Path to Domain Admin)
- Bloodhound is an audit tool as much as an attack tool
  - Helps visualize dangerous trust relationships and misconfigurations
  - Significantly reduces the brute-force effort required to find domain admin
- Very difficult to detect, though tools like GoFetch are very noisy



Bloodhound takes advantage of the security maxim posited by John Lambert of Microsoft: "Defenders think in lists. Attackers think in graphs. As long as this is true, attackers win." It is an automated tool designed to use "graph theory to reveal the hidden and often unintended relationships within an Active Directory environment." [1] Active Directory is immensely complex and tends to gain more complexity as a network evolves. It can be extremely difficult to visualize the consequences of the myriad changes that occur to groups, domains, organization units (OUs), and group policy objects (GPOs) that ultimately affect the trust model. This is why Bloodhound should be considered as much of a blue team (defensive) tool as a red team (attack) tool. Ideally it should be proactively employed in an enterprise by defenders or pen test teams before an attacker gains access to do the same thing. Seeing Bloodhound in action can send a chill down the spine as it effortlessly identifies paths to your most sensitive groups and accounts. The simplified graph shown on this slide is not far off from what can be found on many networks. Because Bloodhound can identify users, groups, and where those users currently have sessions, it presents a step-by-step path of where to dump credentials and where to laterally move next.

Unfortunately, the use of Bloodhound in most environments is very stealthy. To collect data, it uses LDAP requests which are very common in the enterprise. Older versions could sometimes be discovered via Active Directory logs by identifying many near-simultaneous LDAP sessions, but the latest versions of Sharphound (the Bloodhound collector) use cached LDAP connections, limiting the footprint. Attempts have been made to detect it using netflow logs (tracking LDAP sessions), auditing of Directory Service Access in AD logs, and employing honey tokens. EDR tools can potentially use connections to objects like specific named pipes or other behavioral clues. However, just like most of the credential attacks discussed in this section, often the most reliable detection is inappropriate use *after* credential theft occurs. As great examples, there are multiple automation tools that attempt to automate the tasks required to achieve Domain Admin rights, such as GoFetch and DeathStar; these tools can be very noisy.[2,3] GoFetch takes the Bloodhood graph and uses Invoke-Mimikatz and Invoke-Psexec to automate credential theft and lateral movement. Similarly, DeathStar uses PowerShell Empire to enumerate accounts (approximating Bloodhound), perform credential theft, and automate lateral movement. Both methods leave behind a tremendous number of artifacts that we will develop detection capabilities for throughout the next section.

[1] GitHub - BloodHound: https://for508.com/hfi9o
[2] Automating the Empire with the Death Star: getting Domain Admin with a push of a button: https://for508.com/2riwf
[3] GitHub - GoFetch: https://for508.com/xdt0p

This page intentionally left blank.

COURSE RESOURCES AND CONTACT INFORMATION

Here is my lens. You know my methods. -Sherlock Holmes

AUTHOR CONTACT
rlee@sans.org
http://twitter.com/robtlee

ctilbury@sans.org
http://twitter.com/chadtilbury

mpilkington@sans.org
https://twitter.com/mikepilkington

SANS INSTITUTE
11200 Rockville Pike, Suite 200
N. Bethesda, MD 20852
301.654.SANS(7267)

DFIR RESOURCES
digital-forensics.sans.org
Twitter: @sansforensics

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.