3 THE ATOMIC METHODOLOGY

This chapter dives deep into the atomic purple teaming methodology, which allows us to focus exclusively on the activities we can detect. We'll begin by considering the scope of an atomic exercise and the inputs that could influence the test cases we choose to perform. From there, we'll generate an example test suite for the enumeration of an Active Directory group.

Taking the defensive perspective, we'll consider the data points we should collect for each attack we execute. Each test case produces many metrics and other supporting metadata we could capture, many of which depend on the purpose of your exercise, so we'll explore these too.

Finally, we'll consider the practicalities of running a successful atomic purple team, including the necessary prerequisites and the responsibilities of the offensive, defensive, and administrative teams. We'll also touch upon *micro-emulation*, an adversary emulation methodology that seeks to address some deficiencies of the atomic approach and provide a middle ground between it and a full scenario-based exercise.

Applications

In Chapter 11, we'll spend some time considering the broader applications of a purple team's results and business outcomes across an organization's detection-and-response capability. For now, let's consider atomic purple teaming specifically, focusing on the questions we can answer with a well-crafted exercise.

Performance Benchmarking

Atomic purple teaming is uniquely well suited to tracking how your detection capability changes from one exercise to the next. If you keep test cases consistent across exercises, you can track progress over time to demonstrate improvement or regression.

For example, atomic purple teaming can be a great means of showing improvements in configurations; the hardening of endpoints, networks and platforms; and investments made in new tools, log sources, or new rules. You can also ticket test cases and directly reference them when making the case for new log sources and rules (more on this in "Ticketing Systems" on page 263).

While tracking progress in this way can be hugely valuable, also keep in mind that adversary tradecraft is constantly evolving, and some test cases in your exercises may become less relevant as time progresses.

Environmental Comparison

Because they're self-contained, atomic purple teams are easy to repeat across areas of an organization to provide insight into differences in each area's detection capability. This use is particularly relevant to larger, global organizations that have less homogeneity across their systems and processes. For example, atomic purple teams could highlight differences between regional and headquarter offices, mergers and acquisitions, different system builds (Windows 11 versus Windows Server, for instance), virtualized versus on-premise builds, or different logging and monitoring arrangements.

As with performance benchmarking, ensure that the tests from one environment remain relevant for another. If one testing area is a restricted desktop environment, you might want to understand the detection coverage for attempted breakouts to the underlying server, which likely wouldn't be relevant in other locations.

Tooling Evaluation

Atomic purple teaming can also be used to evaluate existing or prospective security tools. For example, a well-crafted test suite can highlight the relative strengths and weaknesses of a various tools or particular configurations.

Your exercise may also reveal the number of tests for which a particular security tool provided telemetry, alerting, and prevention. By focusing on each specific tool in this dataset, you can quickly see the number of attack techniques covered by a given tool, which can be particularly useful for identifying the most valuable elements of your detection toolset. If you procure a new tool or ingest a new log source, rerunning an exercise can enable you to quantify the impact of that investment.

Automation and Regression Testing

Atomic test cases often present prime candidates for automation, a topic we'll discuss in Part II. Automation makes it easy to repeat and scale your purple teams across larger test suites and multiple environments, and enables teams with less experience in offensive tradecraft to re-create attack techniques for analysis.

Automation also presents a great opportunity to perform *regression testing*, which involves regularly running test cases to ensure that capabilities haven't degraded because of unexpected changes or other external factors. Applied to our detection capability, regular atomic purple team testing can ensure that you're still collecting logs where they should be collected, still surfacing alerts, and continuing to block malicious activities.

In its most mature implementation, an automated solution could run random or preselected atomic tests continuously; further automation could validate that expected log entries and alerts are produced in the appropriate systems.

Industry Comparison

For external vendors conducting atomic purple teams across various industry sectors, the data produced by these exercises can provide interesting insight into current trends. This broader view of multiple organizations' logging, alerting, and prevention can answer questions like the following:

- Which industry sectors perform best?
- How does a customer compare to its peers in the industry?
- Which Kill Chain phases are organizations most and least effective at monitoring?
- Which security tools provide the best (and worst) coverage?
- Does security tool *X* produce an out-of-the-box (or custom) alert for attack technique *Y*?

This last point is likely of particular interest to the vendor's red team but can also be valuable to customers wanting to understand the coverage they could reasonably expect from a current or future security solution based on peers' achievements.

Scoping and Dechaining

The Cyber Kill Chain covered in the preceding chapter conceives of an attack as comprising several ordered steps an attacker must complete to meet their objectives. An atomic purple team exercise breaks this end-to-end attack chain into its individual techniques or procedures. This dechaining also removes any dependency between one technique (or test case) and another.

Before you can dechain an attack, you must determine your exercise's scope. For example, you don't necessarily have to consider the full chain in your exercise. Take a look at Figure 3-1. In this attack chain, an adversary sends an email attachment that uses HTML smuggling to deliver an ISO disk image containing LNK and dynamic-link library (DLL) files. Many ransomware attackers have used variations of this attack chain.

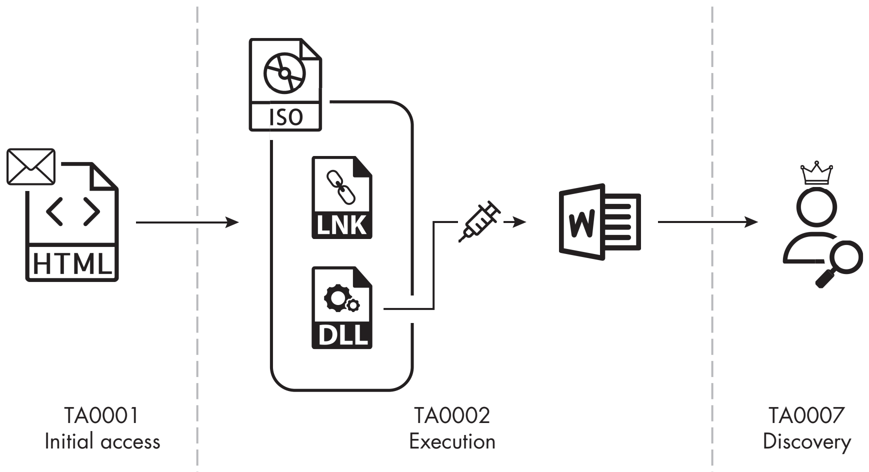

Figure 3-1: An example attack chain with initial access, execution, and discovery phases

Execution of the LNK file launches the DLL via Rundll32 (T1218.011), which subsequently injects an implant into a Microsoft Word process. From this point, an adversary can turn their attention to understanding the environment they find themselves in through the enumeration of high-privileged groups.

Your exercise could focus solely on one area of the attack chain, such as one ATT&CK tactic, and enumerate the offensive activity at that stage. For example, you might evaluate the various other means that an adversary might use to conduct reconnaissance activities in the Discovery tactic (TA0007). This might include evaluating the compromised system and its installed applications, services, and user accounts, as well as its configured system language (a commonly used guardrail to ensure that adversaries, and their malware, don't inadvertently target their own country of origin or those of their allies; see T1614.001).

If you wanted to narrow the scope of your exercise even further, you could focus on just one technique, then enumerate the procedures that could be used to achieve it. Your exercise might simulate various means of achieving process injection (T1055), consider other possible attachment formats, or even just explore the numerous techniques for achieving HTML smuggling that an adversary might use for phishing with an attachment (T1566.001).

These exercises obviously have drastically different scopes, and you'll therefore draw different conclusions from their results. This flexible scoping is a key benefit of the atomic purple teaming methodology, because it enables you to tailor your exercise to the questions you want answered. For example, which types of process injection does your EDR detect? Which forms of HTML smuggling does your mail gateway prevent?

At the narrowest scopes, your exercises might involve just a handful of test cases. Organizations still familiarizing themselves with this methodology or looking for a quick turnaround to validate their exposure to a given threat might find these smaller exercises more manageable.

Inputs

It's important that you execute the right offensive activities to validate your defenses. Let's consider the sources you could use to devise your atomic purple team test cases. While not exhaustive or prescriptive, your sources could include threat intelligence and incident reports, offensive testing outputs, security tooling capabilities, and research.

Atomic test cases should help answer specific questions about the current state of your organization's detection capability, and it's worth keeping these goals in mind as you conceive of a test case. We'll discuss these sources in more detail in Part III, where we'll consider planning purple teams more broadly.

Threat Intelligence and Incident Reports

Digesting threat intelligence and understanding its relevance to the organization you're defending is an entire discipline in itself. Threat intelligence comes in many forms, but it largely falls into one of four high-level categories: strategic, operational, technical, and tactical.

Strategic intelligence provides a broader view of the current and future activities of adversaries. It could include geopolitical or environmental trends, as well as any predicted adversarial reaction to an organization's business activities. This helps inform executive stakeholders and is largely nontechnical in nature.

Operational intelligence focuses on the nature of threat actors, including their motivations and resultant objectives. This helps organizations identify how, why, and by whom they might be targeted, as well as how they might defend themselves, whether as victims of interest or victims of opportunity.

As we explored in "The Pyramid of Pain" on page 51, a detection stack can ingest indicators of compromise (IoCs) to detect and prevent hashes, domain names, IP addresses, sender email addresses, and more. While crucial for detection and response, this *technical intelligence* is less useful from an emulation perspective.

This leaves the last category, *tactical intelligence*, which focuses on the top levels of the pyramid, such as file and network artifacts, tools, and TTPs. You'll find it most useful to acquire details of adversaries' procedural-level activity, which will allow you to plan and execute a faithful re-creation of adversary tradecraft and gain the most accurate understanding of your defensive resilience to it.

Based on your organization's maturity and the resources available, you might be privy to both public and privileged sources of threat intelligence. You may also be a member of an Information Sharing and Analysis Center (ISAC), which can provide great sources of threat intelligence relevant to your industry.

If you must rely on only public sources, you should still find them more than sufficient for emulation material. One notable provider of threat intelligence is the DFIR Report, an organization that publishes high-quality write-ups of adversary activity, including detailed procedures perfect for emulation.

One thing to bear in mind as you're devising atomic test cases is the temporality of threat intelligence. Tradecraft evolves, and what occurs in one documented intrusion might not always reflect an adversary's arsenal at a later date.

Offensive Testing Outputs

Many organizations conduct offensive testing as a key part of their security strategy, whether that includes covert red team exercises, assumed breaches, or purple team engagements. No matter the format, these exercises always highlight defensive deficiencies. Rather than waiting for the next regularly scheduled engagement to check whether you've remediated these deficiencies, you could feed them directly into atomic purple team exercises to validate your new detections or preventative measures.

Further, if your organization's ability to digest threat intelligence and keep tabs on general trends in offensive tradecraft isn't very mature, leveraging the insight of external offensive practitioners and the techniques they use could be invaluable.

Security Tooling Capabilities

Developing an exercise based on the capabilities of your security tooling is an approach uniquely applicable to atomic purple teaming. Using this method, you'll digest the attack coverage that one or more security tools should provide and develop test cases to independently validate these in your environment.

The point isn't just to confirm you're getting what you paid for; all manner of deployment and implementation issues could present themselves, many of which would go unnoticed unless tested for. As an example, say you use a solution that detects identity-based Active Directory attacks. You might include test cases for Kerberos abuse through permutations of techniques like Kerberoasting (T1558.003), Overpass-the-Hash (T1550.002), Golden Tickets (T1558.001), and so on.

Such engagements could produce all kinds of interesting outcomes, including any undocumented edge cases the tools may have and any configuration issues that you could remediate to improve performance. Notably, however, these tests rely on knowledge of your environment or domain to understand the problem space and the possible or relevant attack techniques. If your organization doesn't possess this knowledge internally, you may require a third party to conduct this test for you.

Research

Talented members of the information security community publish an incredible amount of research that advances the field's offensive and defensive disciplines. For example, research could highlight new methods to evade existing Windows endpoint defenses through techniques like Event Tracing for Windows (ETW) tampering, direct or indirect syscalls, and call-stack spoofing. Or the research could detail how specific platforms and technologies used in your organization could hypothetically be attacked and subsequently defended. For examples, take a look at Will Schroeder and Lee Chagolla-Christensen's Active Directory Certificate Services research or Adam Chester's research into the identity provider Okta, included in this chapter's resources.

You could distill this research content into a series of test cases that validate your defensive posture against the attacks outlined. If you're fortunate enough to have an internal research capability, you could use it as the input for your purple team exercises, though this may require a relatively high degree of offensive proficiency to carry out (and an even greater proficiency to produce the research in the first place).

Generating Test Cases

Now that you've considered the scope of your exercise and chosen an input source, you can turn your attention to test-case generation. In this section, we'll walk through an example that focuses on the last stage of the attack chain shown previously in Figure 3-1, and consider how adversaries might discover members of the privileged Active Directory group *Domain Admins* (T1069.002).

Domain Admins is a common target for adversaries, who could uncover its members in several ways. A nonexhaustive list might include *net.exe* binary usage, PowerShell scripting, a third-party application (such as AdFind), and in-memory tooling. Let's consider each of these tools, including how and why we might include them in a purple team exercise. We won't go into too much depth on detection methods here, as we'll save this topic for Part II.

net.exe

The `net` command is a built-in utility in Windows that, among other features, enables the enumeration of local and domain users and groups. Because it is present on the operating system by default (at *C:\Windows\System32\net.exe*), it doesn't require an adversary to install or download any additional files. It's also regularly used in legitimate administrative activity and automation. Reviewing the technique in ATT&CK, you can see procedure permutations drawn from an array of threat actors, including Turla and FIN7. `net` is also included in the Conti playbook mentioned in the previous chapter.

An attacker might run the tool with the following command line arguments to enumerate members of the Domain Admins group:

```
net group "domain admins" /domain
net group "domain admins" /dom
net group /domain "domain admins"
```

The first example uses the basic command syntax, while the second shortens the `/domain` flag and the third reorders the command line arguments. When run, these commands ultimately produce the same result.

An attacker could also take advantage of endless obfuscation opportunities. (You might even want to evaluate these in a separate exercise specific to obfuscation techniques.) The following obfuscated version of this command uses an environment variable to save the Domain Admins group name before referencing it in a `net` command padded with caret (`^`) symbols:

```
Set GROUP= "Domain Admins"
n^e^t g^r^o^u^p %GROUP% /d^o
```

In terms of detection, here is an excerpt from a Sigma rule for general domain group reconnaissance (a link to this rule is provided in this chapter's resources):

```
title: Suspicious Group And Account Reconnaissance Activity Using Net.EXE
id: d95de845-b83c-4a9a-8a6a-4fc802ebf6c0
status: experimental
description: Detects suspicious reconnaissance command line activity on
    Windows systems using Net.EXE
author: Florian Roth (Nextron Systems), omkar72, @svch0st, Nasreddine
    Bencherchali (Nextron Systems)
--snip--
logsource:
    category: process_creation
    product: windows
detection:
 ❶ selection_img:
        - Image|endswith:
              - '\net.exe'
              - '\net1.exe'
        - OriginalFileName:
              - 'net.exe'
              - 'net1.exe'
    selection_group_root:
        CommandLine|contains:
         ❷ - ' group '
```

```
            - ' localgroup '
    selection_group_flags:
        CommandLine|contains:
          ❸ - 'domain admins'
            - ' administrator'
            - ' administrateur'
            - 'enterprise admins'
            - 'Exchange Trusted Subsystem'
            - 'Remote Desktop Users'
            - 'Utilisateurs du Bureau à distance'
            - 'Usuarios de escritorio remoto'
          ❹ - ' /do'
    filter_group_add:
        CommandLine|contains: ' /add'
    selection_accounts_root:
        CommandLine|contains: ' accounts '
    selection_accounts_flags:
        CommandLine|contains: ' /do'
    condition: selection_img and ((all of selection_group_* and not
     filter_group_add) or all of selection_accounts_*)
--snip--
```

This detection checks for processes being created from certain executables (namely, *net.exe* and *net1.exe* ❶), as well as the use of specific command line arguments, at ❷, ❸, and ❹. Recalling the pseudocode representation of detection logic in MITRE's Cyber Analytics Repository in the preceding chapter, this content will likely be familiar. You'll learn more about Sigma in Chapter 7.

While iterating through these procedure permutations might seem like making menial variations of the same command, certain versions might evade detection rules that the others do not.

PowerShell

Offensive uses of PowerShell became incredibly popular in the mid-2010s, in part because of a lack of defensive technologies available to inspect such activity. PowerShell offers many advantages to attackers, including the ability to operate in-memory with "fileless" malware and leverage a wide array of .NET framework APIs to interact with the Windows operating system.

Too many excellent PowerShell tools and frameworks exist to list here, but some notable ones include the following:

PowerShell Empire A fully featured command-and-control framework that includes encrypted communications and numerous post-exploitation features, such as keylogging and screenshot capture

PowerSploit A collection of PowerShell modules for implementing attack techniques across the Cyber Kill Chain, including reconnaissance, persistence, privilege escalation, and more

Nishang A collection of offensive PowerShell modules that includes techniques for payload creation in a variety of formats, lateral movement, and credential access

To enumerate members of the Domain Admins group, you could execute the following PowerShell commands:

```
❶ $Group = [ADSI]"LDAP://CN=Domain Admins,CN=Users,DC=Contoso,DC=com"
❷ $Group.member | ForEach-Object {
      $Searcher = [adsisearcher]"(distinguishedname=$_)"
   ❸ $Searcher.FindOne().Properties.cn
  }
```

This short script creates an Active Directory Service Interfaces (ADSI) object that targets the Domain Admins group based on its `distinguishedName` attribute, `CN=Domain Admins,CN=Users,DC=Contoso,DC=com` ❶. Then, it iterates over the group's members ❷ and prints out their common name attributes ❸.

If you install the Active Directory PowerShell module, you might also run the following cmdlet:

```
Get-ADGroupMember -Identity "Domain Admins"
```

Finally, by leveraging the PowerSploit framework mentioned earlier, you could use its PowerView script, as threat actors have done in the past:

```
Get-DomainGroupMember "Domain Admins"
```

In recent years, Windows has implemented the Antimalware Scan Interface (AMSI) and logging mechanisms like script block logging to detect and prevent PowerShell-based malware.

AdFind

As an alternative to using native software like `net` commands or PowerShell, adversaries could download additional tools to get the job done. As we saw in ATT&CK's software category, this could include malware developed explicitly for nefarious use, or a legitimate tool the threat actor has repurposed.

A prime example of the latter is the free Active Directory query utility AdFind. A myriad of threat actors have used the command line tool (see S0552), to the extent that some antivirus products now flag it as malicious.

We could use AdFind to perform Domain Admin reconnaissance with the following command:

```
AdFind.exe -b "CN=Domain Admins, CN=Users, DC=Contoso, DC=com" member
```

Note the use of the same `distinguishedName` value from the PowerShell ADSI example.

In-Memory Tools

Finally, we could explore procedures that use the feature sets of command-and-control frameworks to execute domain reconnaissance. As defensive improvements reduced the viability and popularity of PowerShell tradecraft, alternative means of executing post-exploitation tradecraft have risen to the fore. Here are three of the most notable alternatives: reflective DLL injection, in-memory .NET execution (often simply referred to by its command alias in Cobalt Strike, `execute-assembly`), and Beacon Object Files (BOFs).

While Cobalt Strike arguably popularized the latter two initially, today the features exist in numerous command-and-control frameworks, which allow you to use all manner of tools and scripts on a compromised host. In relation to the Domain Admins reconnaissance, examples might include the following:

- The use of the Outflank Recon-AD tool as a reflective DLL
- The in-memory execution of Ruben Boonen's .NET tool StandIn
- The execution of the ldapsearch BOF from the TrustedSec Situational Awareness BOFs collection

Yet another alternative might be the deployment of a SOCKS proxy through an existing command-and-control channel, through which network traffic from a local tool, like the Impacket *net.py* script, may be tunneled.

Evaluating Test Suites

Once you've captured your test cases, you can aggregate these into a test suite, such as the one shown in Table 3-1. This test suite highlights several ways that you could perform the relatively simple task of retrieving the members of the Domain Admins group (and shows you how wildly different these procedures can be). This variation should demonstrate why creating a detection to cover one procedure doesn't mitigate the entire ATT&CK technique.

Table 3-1: A Test Suite to Enumerate Members of the Domain Admins Group

Test name	Command	Type
A basic `net` command	`net group "Domain Admins" /domain`	net
A `net` command with a shortened domain flag	`net group "domain admins" /dom`	net
A `net` command with a reordered flag	`net group /domain "domain admins"`	net

An obfuscated `net` command	`set GROUP="Domain Admins"`
	`n^e^t g^r^o^u^p %GROUP% /d^o`
		net

An ADSI searcher script	`$Group = [ADSI]"LDAP://CN=Domain Admins, CN=Users,`
	`    DC=Contoso,DC=com"`
	`$Group.member | ForEach-Object {`
	`    $Searcher = [adsisearcher]"(distinguishedname=$_)"`
	`    $Searcher.FindOne().Properties.cn`
	`}`
		PowerShell

An RSAT Active Directory cmdlet	`Get-ADGroupMember -Identity "Domain Admins"`	PowerShell
A PowerView command	`Get-DomainGroupMember "Domain Admins"`	PowerShell

An AdFind command	`AdFind.exe -b "CN=Domain Admins, CN=Users,`
	`    DC=Contoso, DC=com" member`
		AdFind

The use of StandIn	`execute-assembly /tools/StandIn.exe --group`
	`    "Domain Admins"`
		In-memory .NET execution

The use of ldapsearch	`ldapsearch "CN=Domain Admins" member`	BOF
The use of Recon-AD	`Recon-AD-Groups Domain Admins`	Reflective DLL
The use of SOCKS with Impacket *net.py*	`socks 8080` (on Cobalt Strike beacon) `proxychains python net.py user:pass@dc group -name` `    “Domain Admins”` (on attacker host command line)	SOCKS and Impacket

While you’ve collected a pretty varied list of procedures at this stage, you’ve no doubt left out a multitude of other tools and scripts. So, you might now want to evaluate your coverage by considering tool and command permutations used by relevant threat actors or red teams, as well as procedures presented by new offensive research.

Capability Abstraction

A great way to triage new tools and procedures for inclusion in your purple team exercises is by applying Jared Atkinson's *capability abstraction*. This concept highlights that although the numerous offensive tools that could be used for a task may use different coding languages and have different ways of executing, they often overlap in the fundamental ways they achieve their capabilities.

An example in our case study might be the use of `net` commands and the SOCKS-tunnel Impacket *net.py*. While these attacks use completely different means of execution, both achieve Active Directory enumeration through requests made to domain controllers by using the Security Account Manager Remote protocol (SAMR). If you develop a detection capability at the network level or via agents deployed on domain controllers, the attacker's manner of employing the technique on the endpoint makes little difference to your ability to detect the requests being made.

A similar overlap exists with test cases like the ADSI searcher PowerShell command and the use of StandIn, both of which use the .NET framework as well as classes and functions in the *System.DirectoryServices* namespace to send Lightweight Directory Access Protocol (LDAP) queries to a domain controller.

This overlap between tools and techniques has both offensive and defensive implications. From an offensive perspective, if you're striving to comprehensively evaluate your detection coverage for the reconnaissance of domain group members, you should be aware that the use of a different tool doesn't necessarily correspond to a different action taking place, though each may introduce tool-specific artifacts (like predictable files, hashes, and user agents).

From a defensive perspective, capability abstraction might sound a lot like the Pyramid of Pain. Depending upon where in the pyramid you've developed your detections, attackers may be able to evade these by simply changing their tool, while detections for the LDAP- or SAMR-based requests might present a bottleneck that requires attackers to change their fundamental technique. Thus, it's worth keeping capability abstraction in mind when developing your test suites.

Similarly, the ever-present challenge remains of reducing false positives and more accurately determining malicious intent. If our detection logic for Domain Admins group enumeration operates at a network level, we might lack the ability to discern actions performed as part of legitimate administrative activity from commands spawned from an established implant.

Attack Sophistication

Another point to bear in mind is the relative sophistication of the offensive procedures you're evaluating. With the numerous ways that a given technique can materialize, some are undoubtedly simpler to execute than others.

As an exercise, try taking the four test-case focus areas we've considered and plotting them onto a pyramid of sophistication, as shown in Figure 3-

2, where the prevalence of the procedure decreases as the sophistication increases.

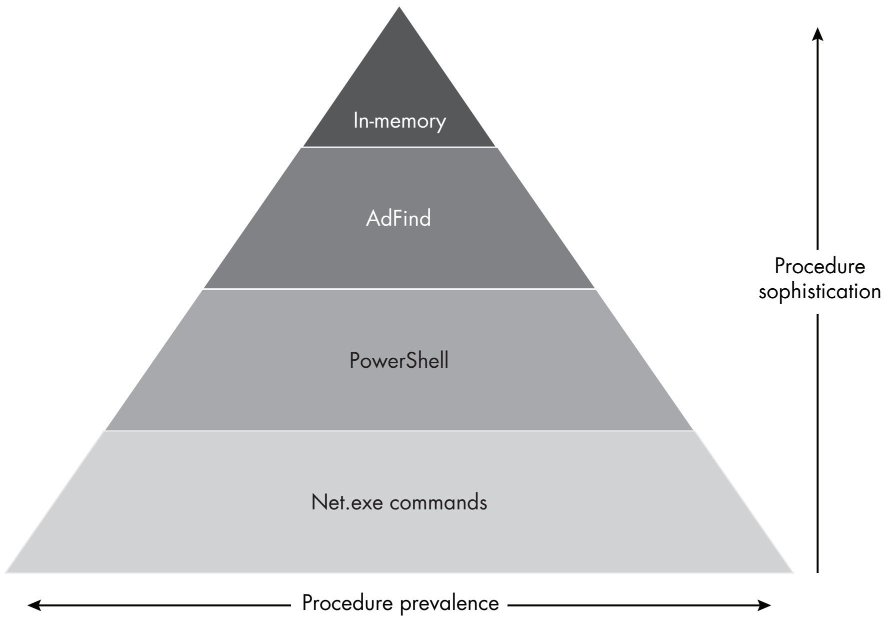

Figure 3-2: Attack techniques arranged in a pyramid of sophistication and prevalence

Based on your experience, you might disagree with the ordering of the pyramid. Still, it should allow you to take away a few key points.

First, the ability to detect a procedure at the top layer of the pyramid doesn't guarantee that you've adequately covered the procedures beneath it. This is a basic tenet of layered detection, or *detection in depth*, and highlights the necessity for detection across multiple layers of the Pyramid of Pain.

Second, you're less likely to detect the higher sophistication attacks, which might explain why they're considered less prevalent. By contrast, you'll find countless threat actor examples of `net` commands used for reconnaissance. A red teamer might scoff at using these latter procedures in an exercise, but having validated coverage for all activities is essential for comprehensive coverage.

Data to Capture

Before you execute your suite of atomic tests, consider the key data points you'll want to gather about each attack you launch. There isn't a prescriptive way to do this, but in my experience, the three most important data points are telemetry, alerting, and prevention.

Put simply, you should ask yourself the following: Are there logs for the activity? Did it trigger an alert? And finally, was the activity blocked? Often you'll find it hard to answer these questions with a binary yes or no and may instead require more nuance. Further, when you start considering the business applications for the data you collect, you'll likely come up with other data points to take into account. For example, you may want to record which security tools or event sources performed the logging, alerting, or blocking of a given test case so you can understand which parts of your detection stack are most or least useful.

Either the offensive team or the defensive party can capture the results, depending on the exercise's structure and available expertise. Generally, however, the ideal arrangement is for the defensive team to interrogate its own data. This task offers a valuable experience for analysts, who many encounter alerts or hunt for data points they don't see every day.

Nonetheless, I've been part of numerous purple team exercises that provision the offensive team with access to detection platforms and task them with capturing the results themselves, particularly in the case of external vendor exercises. This is most common when the SOC or monitoring team lacks the knowledge or bandwidth to facilitate results capture.

While we won't discuss it in detail here, the DeTT&CT framework mentioned in Chapter 2 offers one way of standardizing the scoring of data and detection quality. In Chapter 11, we'll use VECTR, a great way to capture these results.

Telemetry

When determining whether telemetry exists for a given test case, you should evaluate two key attributes: whether the log is of high fidelity and whether the log is centralized.

In this context, *fidelity* refers to whether the telemetry provides an accurate and complete representation of the attack that has taken place. Depending on the nature of the test case, this could be a log of network connections made, commands executed, or resources accessed. Fidelity can vary significantly depending on the log source and attack in question. Ultimately, you'll need to use your judgment (or that of the defensive team) to determine whether the logs available detail who did what and when.

Log centralization refers to the aggregation of security events such that monitoring teams have access to them and could turn them into alerts (if these don't already exist). If attack activity took place on a corporate workstation, for example, security tools like *System Monitor* (*Sysmon*) might capture rich telemetry about the commands being executed. However, if these logs aren't forwarded to a centralized location and queried by analysts, you'd have no indication that an attack was taking place.

Log centralization doesn't necessarily mean that every log gets ingested into a single system, just that they're all available to the monitoring team. Because EDR solutions generate significant amounts of data and typically have their own log storage and querying capabilities, it's common to see some or all of this data kept separately from the other logs ingested into a SIEM platform.

Another consideration is the timeliness of these logs. If a log arrives several hours after the attack has taken place, you could argue that it's too late for that log to be actionable and that the answer to "Are there logs for this activity?" is "No."

Alerts

The attributes of fidelity and centralization apply to alerting too, whether you're evaluating an out-of-the-box security solution or custom detection logic. In terms of fidelity, alerts firing as a result of a test-case execution and alerts that correctly notify the SOC of the activity performed aren't necessarily the same thing. Be on the lookout for alerts that are ambiguous or mislabeled.

Another element to consider is which aspect of the attack the alert is detecting. Say you're testing an executable that is uploaded to an endpoint and used to perform password spraying (T1110.003). If an alert fires for a suspicious file in the environment with no mention of password spraying or other indication of its malicious use, does that constitute an actionable alert for this test case?

To answer this question, consider the Pyramid of Pain and capability abstraction. If a relevant threat actor uses the same executable, this alert gives us the valuable insight that we'd likely detect its use based on a hash or signature. But if we want to inflict the greatest pain on adversaries, we should also develop detections that focus on the password-spraying technique itself, regardless of the specific tool used to achieve it.

It's worth bearing in mind that atomic purple teaming doesn't cater well to some methods of detection and alerting. Scheduled scans, such as those performed by EDRs to look for suspicious memory artifacts, can be resource intensive and may run only periodically. If your test cases include variations of process injection that install a persistent agent in memory, they might bypass point-in-time alerting but be picked up later by one of these scans. Returning to the concept of timeliness, you'd have to decide how to account for these alerts in your results.

Detection methods such as risk scoring, detection aggregation, and user and entity behavior analytics (UEBA) also pose problems for this style of purple teaming. You've intentionally sacrificed the realism and typical chronology of an attack chain to execute atomic test cases, performing techniques out of order or running through many permutations of a given technique when an adversary might perform only one. For detection systems that aggregate alerts or analyze pattern-of-life, this high volume of rapid-fire test cases could artificially light up the platforms like a Christmas tree. As a result, the tools might scrutinize offensive techniques performed later in the exercise far more closely than they otherwise would.

If this situation arises, you could ignore this aspect of the detection stack and focus on the logs and alerts that fire from other systems. Otherwise, you should abandon the atomic approach entirely and perform the exercise in a scenario or micro-emulation-based format instead.

Prevention

Ideally, you'll be able to prevent the attacks you emulate in your tests. Restricting what an adversary can do at each stage of their attack chain reduces your attack surface, not to mention playing a part in reducing the volume of alerts for the SOC to investigate.

In this context, prevention can occur directly or indirectly. For example, if you run test cases on an endpoint, an antivirus or EDR solution might directly prevent the offensive activity. A matched file hash, signature, or some behavioral logic might lead the security solution to take direct action against you—for example, blocking your activity or terminating the process that attempted to perform it.

Indirect prevention could occur as a result of hardened endpoints, networks, or other platforms to preclude the possibility of carrying out the offensive activity. Generally, this occurs as the result of a broader attack-surface reduction strategy and not as a targeted response to the test-case activity.

Good examples of this include the following:

- Network segmentation that restricts connectivity between neighboring hosts or network zones, inhibiting discovery activity or lateral movement
- Endpoint solutions such as Windows Defender Application Control, which dictate the applications and code that can run, and in which scenarios

As when categorizing alerts, you can classify preventative measures according to the levels of the Pyramid of Pain, with antivirus blocking a specific tool or file, and comprehensive network segmentation preventing lateral movement techniques regardless of the tools chosen.

Execution

Having developed an atomic test suite that meets your needs, you can turn your attention to the practicalities of planning and carrying out the exercise. In this section, we'll consider the activities that need to take place prior to executing your attacks, and the metadata that could be captured for future analysis.

We'll consider the key elements to a well-planned purple team exercise more broadly in Chapter 12.

Preparing Defenses

More than once, I've happily worked my way through a suite of test cases only for the SOC to notify me that an incorrectly provisioned endpoint agent sent no logs to the SIEM. For this reason, it's always worth double-checking that you've properly configured all relevant security solutions. Make sure your EDR and antivirus agents are up-to-date with the latest rules, signatures, and appropriate policies, and check that logs are being forwarded to the proper locations.

A simple way to perform a preflight check is to drop an EICAR file to disk. Developed by the European Institute for Computer Antivirus Research (EICAR) and the Computer Antivirus Research Organization (CARO), an EICAR file is a benign file intentionally signatured by antivirus vendors, providing a safe way to test that systems are operational without requiring an end user to handle true malware. Some vendors also have their own, custom behavioral or file-based signatures that you might want to leverage to confirm you're all set. Naturally, any issues that arise from these checks could be findings in their own right!

If you've provisioned some allowlisting to facilitate testing, you should validate it at this stage too. This could include confirming that your command-and-control channels were added to web proxies and that antivirus exclusions were put in place for any tools you intend to use.

Ordering Test Cases

As we've already explored, the self-contained nature of atomic test cases means you generally don't have to execute them in any specific order. If you're performing a test suite that covers various ATT&CK tactics or Kill Chain phases, you might want to group test cases in these ways. Alternatively, if you're executing a mix of test cases through one or more command-and-control channels on a host, you might want to group by channel so you're not jumping between different systems.

External requirements might also influence your test-case ordering. For instance, certain systems and platforms (or the teams that manage them) may be available for specific periods only. The SOC might have preferences too, and you could reduce the number of times defenders must switch security platforms if you group tests in a certain way. For example, you could focus on initial access tests that traverse the same mail gateways or lateral movement tests that a network monitoring solution should pick up.

If you're performing your test cases on more than one host for comparison purposes, it might make sense to run each test on all relevant platforms before moving onto the next.

Capturing Metadata

Once you've ordered your test cases, you can begin testing. As you progress through your test suite, it's important to accurately track your activities—not just to capture the exercise's results but also to easily resolve any queries or issues. Someone might want to know who executed a given test case or which tool you used at a given time. Let's consider the metadata you might want to capture in addition to the telemetry, alerting, and prevention statistics we spoke about earlier:

- Who ran the test (the name of an individual, a username, or another identity)

- The hostname from which the test was executed, or another identifying ID, such as the Amazon Resource Name (ARN) of an AWS resource
- The targeted assets (for example, a neighboring host targeted for a port scan)
- The specific commands or tools run, including command line arguments, file hashes, and web API calls made
- The outcome of the test from the attacker's perspective, which can be useful for validating the SOC's understanding of events

You could capture these results as you complete each test case or asynchronously, to fit the availability of the SOC.

Plotting Results

By plotting telemetry, alerting, and prevention scores on a graph, as in Figure 3-3, you can understand your coverage and better fill any gaps. Depending on the scope of the exercise, you might want to create a graph for each Kill Chain phase, ATT&CK tactic, or defensive control, or as an overall evaluation.

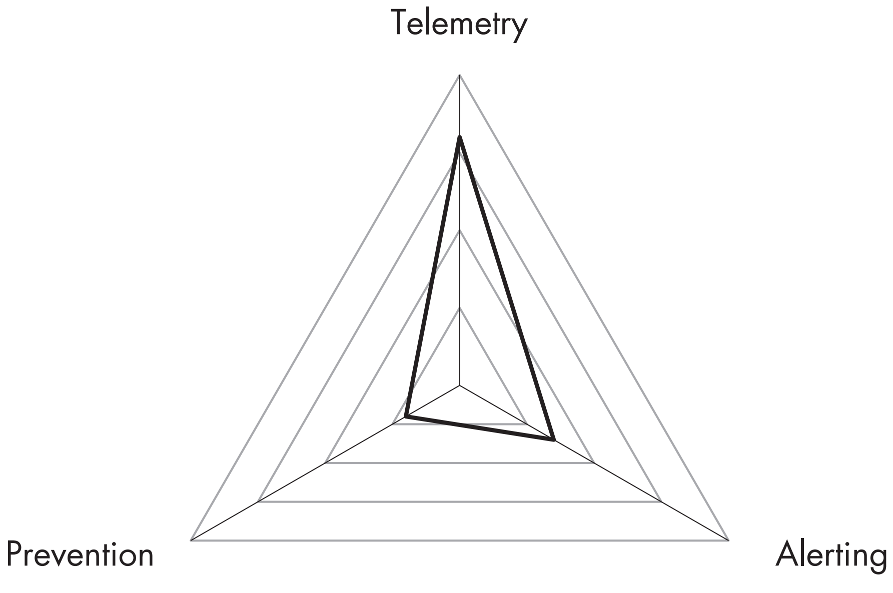

Figure 3-3: Atomic purple team results plotted with a strong telemetry score

This graph shows strong telemetry coverage but little in the way of alerting and prevention, suggesting that the building blocks to develop alerts exist but haven't yet been operationalized. Results like this are common from an immature SOC. The organization might have invested heavily in its logging infrastructure but not yet capitalized on the resultant data, or else it lacks the necessary awareness of offensive tradecraft or detection engineering. There could also be a deficiency in tooling that prevents custom rules from being developed or a contractual arrangement with an MSSP that leaves management of alerts to a third party.

Figure 3-4 shows an alternative outcome. It reveals a strong preventative performance but deficiency in both telemetry and alerting.

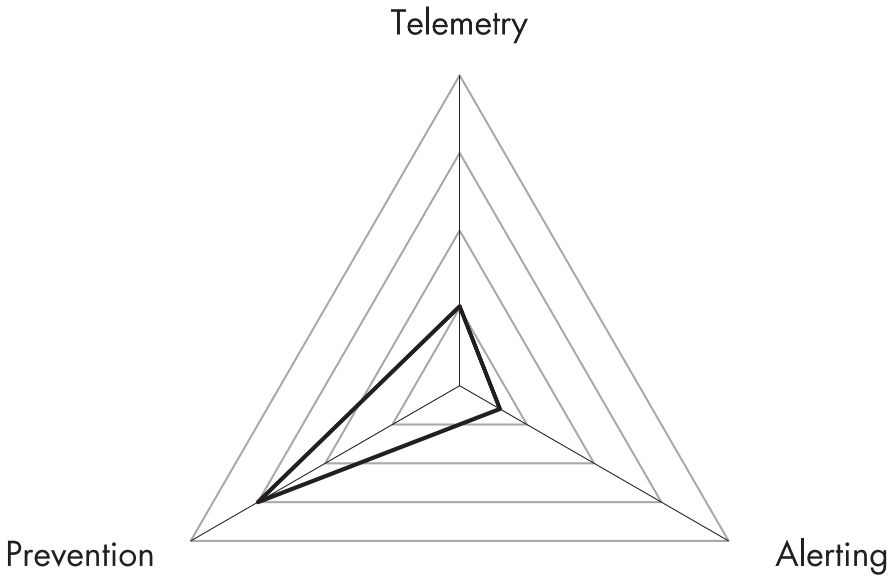

Figure 3-4: Atomic purple team with a strong prevention score

Such an environment may restrict the techniques an adversary can successfully employ, but at the same time, little of their activity will alert the SOC to their presence. I've typically come across results like these in purpose-built environments, such as cloud workloads that have implemented principles of least privilege and good network design but few detections.

You could also plot the three values across each attack phase, as in Figure 3-5.

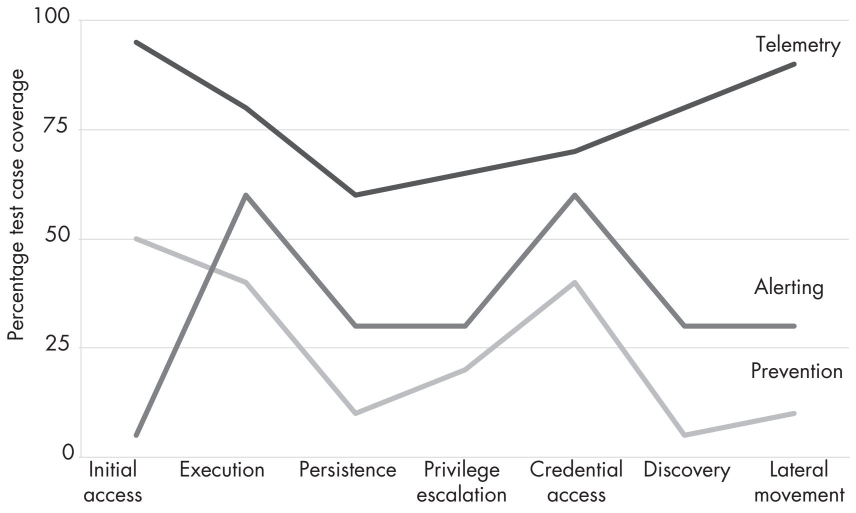

Figure 3-5: Atomic purple team results plotted on a graph across multiple ATT&CK tactics

Now you can see that the environment has good telemetry and prevention coverage for initial access (potentially the result of well-configured web and mail gateways) and strong alerting for execution and credential access, but reduced scores for persistence and discovery. Telemetry remains generally strong throughout the attack chain.

When you evaluate results in this way, however, you might not be able to easily see the places where you're failing to detect and prevent test cases for known adversary techniques. Comparing the percentage of coverage of the telemetry and alerting allows you to determine the residual alerting potential for a given attack-chain phase. Using the same exercise results as in Figure 3-5, you could plot a subset of attack-chain phases in the alternative way shown in Figure 3-6. Here, you can clearly see the difference between telemetry and alerting at each stage.

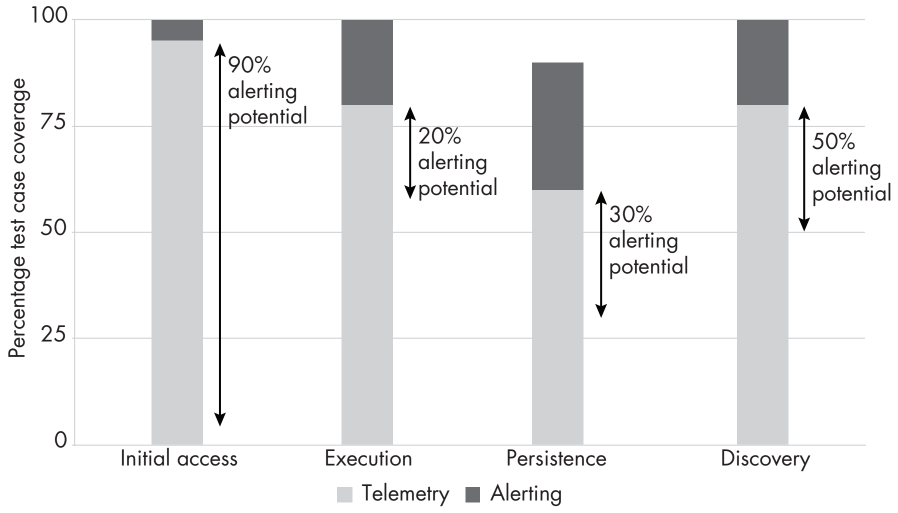

Figure 3-6: Capturing the alerting potential

In some cases, these gaps represent opportunities to improve alerting and get more from the high-fidelity telemetry you already have, though if you have telemetry coverage for only 60 percent of test cases (as in the case for the persistence phase in Figure 3-6), you'll likely need to find additional log sources to increase this percentage before you can substantially improve your alerting potential.

Realizing alerting potential has two other challenges. First, you can't always translate telemetry into alerts. Consider the use of AdFind in the test-case generation we discussed earlier. It's relatively easy to develop rules for the AdFind executable and its command line usage, but if AdFind is used ubiquitously by IT admins across your environment, you might not have the ability to discern malicious usage from legitimate ones.

The second and arguably most important challenge is the need to prioritize alerts. One of the most dangerous traps to get into when atomic purple teaming is adopting a "Whac-a-Mole" mindset when it comes to remediation. For example, Figure 3-6 shows that the most significant alerting potential exists in the Initial Access phase, but it's probably not feasible for the SOC to receive and triage an alert for every inbound email, web download of a blocked file type, or matched antivirus signature.

Similarly, compare an alert for the `whoami` command with the detection of a DCSync attack (T1003.006). While the former has applications for detecting privilege escalation or web-shell activity, detection of the latter (as we'll explore in Chapter 10) would likely be of the highest priority to a SOC monitoring an Active Directory environment.

The key takeaway is that your test suites are rarely "completable" when it comes to telemetry and alerting. The SOC's mandate is to detect every attack, not every technique, and setting objectives to maximize alert coverage can be detrimental to the SOC's overall performance (more on this in "Key Performance Indicators" on page 300).

Micro-Emulation

We've highlighted a key shortcoming of the atomic purple teaming approach: its inability to properly test some forms of alerting logic, including alerts that result from multiple actions happening in a specific sequence and time frame. To test these detections, we need to execute multistep attacks, though we don't necessarily need to proceed through an entire attack chain. In other words, we need something between an atomic and a scenario-based exercise.

In 2022, the Center for Thread-Informed Defense (CTID) and several industry partners published the details of an alternative purple teaming methodology called *micro-emulation*. This methodology aims to get the best of both worlds by providing compound behaviors that test detections for multistep activities while remaining easily automated and low effort for teams to perform.

Upon release, CTID provided nine micro-emulation plans that organizations could immediately carry out in their own environments. These plans targeted a range of notable adversary behaviors, including the following (links can be found in this chapter's resources):

- Web-shell usage
- Fork-and-run (a popular execution technique used by C2 frameworks, such as Cobalt Strike)
- Active Directory enumeration
- Launch of a payload delivered via phishing

Considering the last entry here as an example, this emulation plan automates the mounting of an ISO disk image on an endpoint, which is then followed by the execution of a script file stored within it.

Use of container formats like ISOs has historically been a popular choice for several malware strains, including IcedID, QakBot, and Bumblebee (more info can be found in the links in this chapter's resources).

Historically, disk image files were of particular interest for malware delivery because the Mark of the Web (MotW, T1553.005) was being improperly applied to their contents. This bypassed several built-in Windows protections that would have otherwise impeded malware infection—something that Microsoft patched in late 2022.

From a detection perspective, this emulation plan enables the evaluation of alerting that triggers upon script content being launched from a mounted disk image.

Wrapping Up

This chapter introduced the first of the test methodologies covered in this book: atomic purple teaming. You explored how to deconstruct end-to-end attack chains to develop, validate, and maintain detection coverage at each stage.

The versatility of atomic purple teaming means you don't have to stop there, though. A broad range of sources could serve as inputs to help shape your exercises, enabling you to develop test suites that focus on specific security tools, research, or the outputs of other offensive testing.

To demonstrate this, we developed an example test suite focusing exclusively on the enumeration of the Active Directory Domain Admins group (T1069.002), including tests using native commands, PowerShell, third-party tools, and command-and-control frameworks. Throughout this process, we saw how concepts like capability abstraction and the Pyramid of Pain interact to inform the way we think about test cases.

Atomic purple teaming can provide a host of useful data points, the most important being which attacks were logged, alerted upon, and prevented. In this chapter, we also considered other metadata, like who executed the test case and the specific commands run—all useful information for the SOC when trying to find evidence of the activity.

We then stepped through some of the practicalities of organizing and conducting atomic purple teams, including the relevant approvals, collaboration between teams, and checks for ensuring that exercises run as smoothly as possible. Finally, we looked at analyzing results data to highlight certain elements of detective capability, like alerting potential, and apply our findings to performance benchmarking, tracking return on investment, and comparing security tools.

Resources

Abrams, Lawrence. "Microsoft Fixes Windows Zero-Day Bug Exploited to Push Malware." BleepingComputer, November 22, 2022. *https://www.bleepingcomputer.com/news/microsoft/microsoft-fixes-windows-zero-day-bug-exploited-to-push-malware/*. An article covering Microsoft's patch of the Mark of the Web bypass for ISO disk images.

Atkinson, Jared. "Capability Abstraction." SpecterOps, Feb 6, 2020. *https://posts.specterops.io/capability-abstraction-fbeaeeb26384*. Introducing the concept of capability abstraction.

Australian Signals Directorate. "Detect and Prevent Web Shell Malware." June 9, 2020. *https://media.defense.gov/2020/Jun/09/2002313081/-1/-1/0/CSI-DETECT-AND-PREVENT-WEB-SHELL-MALWARE-20200422.PDF*. Guidance on the detection and prevention of web shell malware.

Bohannon, Daniel. "DOSfuscation: Exploring the Depths of cmd.exe Obfuscation and Detection Techniques." Mandiant, 2018. *https://cloud.google.com/blog/topics/threat-intelligence/dosfuscation-exploring-obfuscation-and-detection-techniques/*. Examples of command line obfuscation techniques.

Center for Internet Security. "Surge of QakBot Activity Using Malspam, Malicious XLSB Files." Accessed January 12, 2025. *https://www.cisecurity.org/insights/blog/surge-of-qakbot-activity-using-malspam-malicious-xlsb-files*.

Another example of a disk-image-based initial compromise to deliver QakBot malware.

Center for Threat-Informed Defense. "Micro Emulation Plan: Active Directory Enumeration." GitHub. Accessed January 12, 2025. *https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/9786a3297c855ea8dfa6c321befa397473b32f41/micro_emulation_plans/src/ad_enum*. A micro-emulation plan replicating Active Directory enumeration through LDAP queries, Windows APIs, and built-in executables.

Center for Threat-Informed Defense. "Micro Emulation Plan: Named Pipes." GitHub. Accessed January 12, 2025. *https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/9786a3297c855ea8dfa6c321befa397473b32f41/micro_emulation_plans/src/named_pipes*. A micro-emulation plan replicating the popular C2 framework technique fork-and-run.

Center for Threat-Informed Defense. "Micro Emulation Plan: User Execution." GitHub. Accessed January 12, 2025. *https://github.com/center-for-threat-informed-defense/adversary_emulation_library/blob/9786a3297c855ea8dfa6c321befa397473b32f41/micro_emulation_plans/src/user_execution/README_user_execution.md*.A micro-emulation plan replicating user-driven execution of an initial access payload that could be delivered via phishing.

Center for Threat-Informed Defense. "Micro Emulation Plan: Web Shells." GitHub. Accessed January 12, 2025, *https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/9786a3297c855ea8dfa6c321befa397473b32f41/micro_emulation_plans/src/webshell*. A micro-emulation plan replicating web shell activity.

Champion, Alfie. "HTML Smuggling: Recent Observations of Threat Actor Techniques." delivr.to, January 6, 2023. *https://blog.delivr.to/html-smuggling-recent-observations-of-threat-actor-techniques-74501d5c8a06*. Examples of the many techniques for performing HTML smuggling.

Chester, Adam. "Okta for Red Teamers." TrustedSec, September 18, 2023. *https://trustedsec.com/blog/okta-for-red-teamers*. A collection of post-exploitation techniques targeting components of the identify provider Okta.

Cunningham, Mike, and Jamie Williams. "Ahhh, This Emulation is Just Right: Introducing Micro Emulation Plans." September 15, 2022. *https://medium.com/mitre-engenuity/ahhh-this-emulation-is-just-right-introducing-micro-emulation-plans-7bf4c26451d3*. Release of the micro-emulation framework from the CITD.

DFIR Report. "Bumblebee: Round Two." September 26, 2022. *https://thedfirreport.com/2022/09/26/bumblebee-round-two*. Bumblebee malware infection achieved via LNK and DLL files contained in an ISO disk image.

DFIR Report. "Malicious ISO File Leads to Domain Wide Ransomware." April 3, 2023. *https://thedfirreport.com/2023/04/03/malicious-iso-file-leads-to-domain-wide-ransomware/*. Demonstration of IcedID malware delivery via an ISO disk image file, designed to bypass Mark of the Web.

EICAR. "What Is the EICAR Test File?" Accessed February 29, 2024. *https://www.eicar.org/download-anti-malware-testfile/*. An EICAR test file.

McGrath, Brandon. "Execution Guardrails: No One Likes Unintentional Exposure." TrustedSec, August 6, 2024. *https://trustedsec.com/blog/execution-guardrails-no-one-likes-unintentional-exposure*. Operational considerations and technical implementation details of environmental keying and execution guardrails

Microsoft. "How the Antimalware Scan Interface (AMSI) Helps You Defend Against Malware." August 23, 2019. *https://learn.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps*. An overview of the architecture and impact of the Antimalware Scan Interface.

Microsoft. "PowerShell Loves the Blue Team." June 9, 2015. *https://devblogs.microsoft.com/powershell/powershell-the-blue-team/*. An overview of detection-and-prevention mechanisms introduced by Microsoft in Windows PowerShell version 5 and onward.

Mudge, Raphael. "Cobalt Strike 3.11—The Snake That Eats Its Tail." Cobalt Strike, April 9, 2018. *https://www.cobaltstrike.com/blog/cobalt-strike-3-11-the-snake-that-eats-its-tail*. Introducing `execute-assembly` to Cobalt Strike.

Mudge, Raphael. "Cobalt Strike 4.1—The Mark of Injection." Cobalt Strike, June 25, 2020. *https://www.cobaltstrike.com/blog/cobalt-strike-4-1-the-mark-of-injection*. Introducing BOFs to Cobalt Strike version 4.1.

Roth, Florian, and omkar72, @svch0st, and Nasreddine Bencherchali. "proc_creation_win_net_groups_and_accounts_recon.yml." GitHub. Accessed January 12, 2025. *https://github.com/SigmaHQ/sigma/blob/fad4742996c55d8d4663e611f84877a2b741dc46/rules/windows/process_creation/proc_creation_win_net_groups_and_accounts_recon.yml*. A Sigma rule to detect the enumeration of high-value groups like enterprise and domain administrators via the built-in `net.exe` executable.

Schroeder, Will, and Lee Chagolla-Christensen. "Certified Pre-Owned." SpecterOps, June 22, 2022. *https://specterops.io/wp-content/uploads/sites/3/2022/06/Certified_Pre-Owned.pdf*. A whitepaper covering the extensive abuse potential of misconfigured ADCS.

Yardley, Michael. "Breaking Out of Citrix and Other Restricted Desktop Environments." Pen Test Partners. June 6, 2014. *https://www.pentestpartners.com/security-blog/breaking-out-of-citrix-and-other-restricted-desktop-environments/*. Techniques for breaking out of Citrix environments.