Embracing offensive tooling: Building detections against Koadic using EQL

By Daniel Stepanic

Published: 2022-06-01 · Archived: 2026-04-05 16:13:54 UTC

This year at BSidesDFW, my local security conference, I highlighted a continuing trend of adversaries using open source offensive tools. The talk reviewed one of these post-exploitation frameworks named Koadic and walked through different ways defenders can build behavioral detections through the use of Event Query Language (EQL). In this post, I wanted to review this research by providing background into Koadic and its features, why it's relevant, and then dive into some EQL examples where I will share different detection strategies against the Koadic framework.

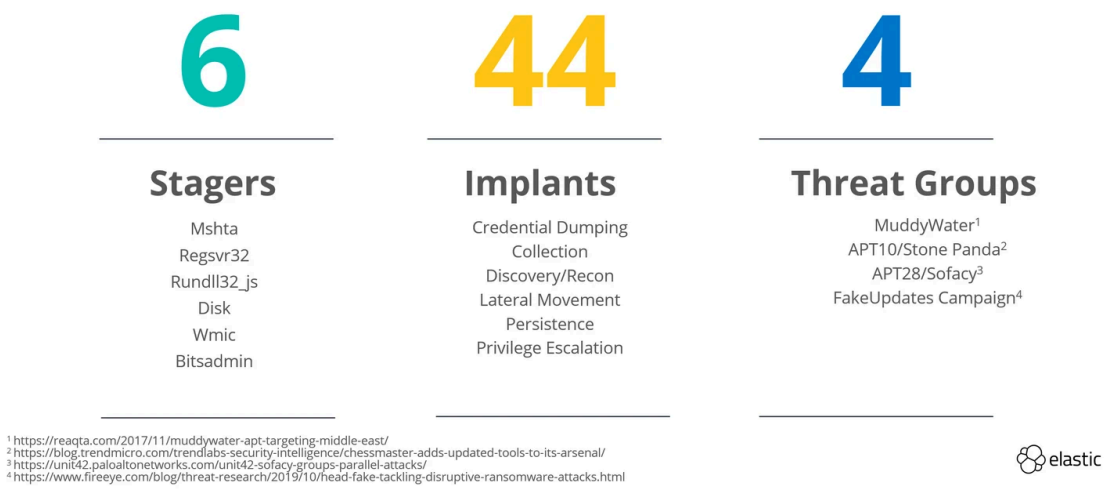
Adversaries continue to adopt open source attack frameworks as part of their toolset. By using these off-the-shelf tools, attackers are able to complete their objectives while reducing their development costs and present attribution problems for incident responders. These tools challenge traditional investigative techniques by creating the idea of plausible deniability and leave fewer toolmarks that can be traced back to an adversary. Even with strong threat intelligence capabilities and proper defensive visibility, it's not always an easy task to differentiate red team operations from real adversary behavior — especially in the early phases of an attack.

As defenders, we are required to actively monitor offensive open source projects. These projects serve as canaries in a coal mine, giving us an opportunity to gain insights into new attacker tradecraft. Not only does this get us into an attacker mindset, but all the code is freely available for emulation and review. Some different ways to get value from dissecting these tools can be through validating your detection capabilities, generating new analytics, developing threat hunting hypotheses, or by simply transferring knowledge around an underlying behavior.

Why Koadic?

Koadic is a great candidate to demonstrate behavior-based detections due its interesting way of leveraging technologies built into the Windows OS — such as Windows Script Host (WSH) and the Component Object Model (COM). COM and WSH fall into the general category of "living off the land" tools, which allow adversaries to proxy the execution of their toolset through built-in Windows programs.

This can be very desirable from an attacker's standpoint, as it allows them to blend in better within the network while producing less of a forensic footprint — rather than dropping and executing a file on disk. COM and WSH are particularly attractive to attackers because of a lack of robust, built-in logging capabilities in contrast to tools like PowerShell (for which Windows has added extensive logging capabilities in newer versions). Koadic is good enough for mature threat groups such as APT28 and has received frequent updates over the last couple years.



Koadic overview

Koadic overview & features

EQL

Some of the more exciting parts of my job as a practitioner come when writing behavioral detections using EQL. This process brings out my inner detective skills that I find to be fulfilling and challenging at the same time. The language enables practitioners to go beyond matching static values such as Indicators of Compromise (IoCs) to a much more comprehensive and flexible way to detect adversary behaviors. With the ability to leverage features like time-bound sequencing or track process lineage, more options are opened up to us as defenders to build reliable and lasting detections.

I find this to be rewarding, as I can directly apply my previous SOC work experience around host-based forensics into a much more dynamic detection that holds up against the latest attacker tradecraft. The best part is that EQL has a very simple syntax with a short learning curve, so if you aren't able to adopt EQL today, hopefully the logic within these queries can be applied to your current solution.

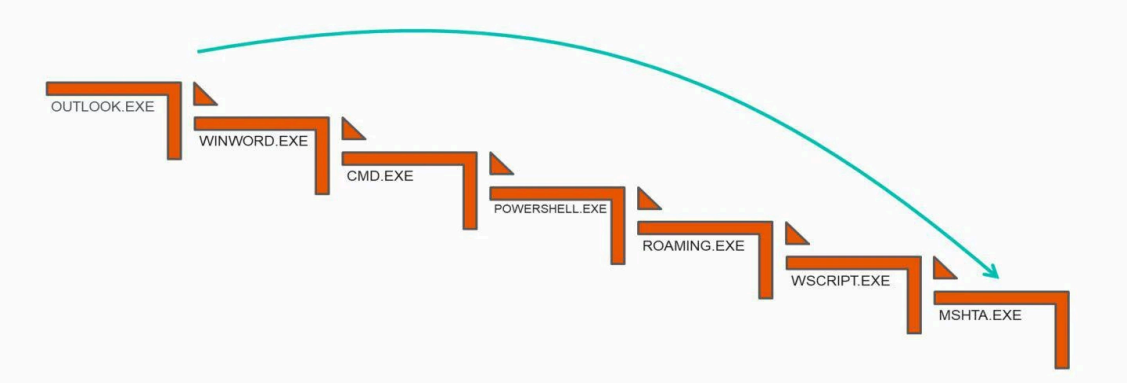
For a quick recap into the language itself and its core capabilities, I recommend reviewing our previous blog post: Introducing Event Query Language. In summary, EQL is a schema-independent and OS-agnostic language built for real-time detection with stream processing. It supports multiple-event behaviors and also offers post-processing commands used to analyze large amounts of data. With that background, let's start diving into some of the different ways to detect Koadic using EQL.

Initial access

Spearphishing Attachment (T1193)

Validating parent-child process relationships continues to be a fruitful technique for hunting and building detections. As attacker activity becomes more evasive, however, we also have a need for flexibility as defenders. This first example shows off the value of tracking process ancestry using EQL's descendant function, which lets us maintain state and track full process genealogy. This is an important concept because it goes beyond the typical parent-child process paradigm that many languages are limited to.

By tracking further descendant processes, defenders have the ability to follow process chains infinitely down the process tree. This provides more freedom in how we can express suspicious behavior together while also allowing tighter controls around process ancestry.



Descendant process tree

Descendant process tree visualization (APT28)

This process chain comes from a sample reported by Palo Alto Networks in June 2018 associated with APT28 activity. In this EQL expression, we are monitoring all descendant processes of our email client (Outlook.exe) and only looking for new process creation events tied to Mshta. This allows us to focus on the initial attack vector (Spearphishing email) and then filter on Windows programs being used to execute attacker code. This is a great foundation for strong analytics — if we wanted to create something more robust, we could build out a longer array of cohorts associated with initial compromise stages, as well as add the entire Microsoft Office suite as descendants.

Initial access and execution - spearfishing

Initial access & execution - Spearphishing example

EQL query:

```
 process where process_name == "mshta.exe" and descendant of
[process where process_name == "outlook.exe"]
```

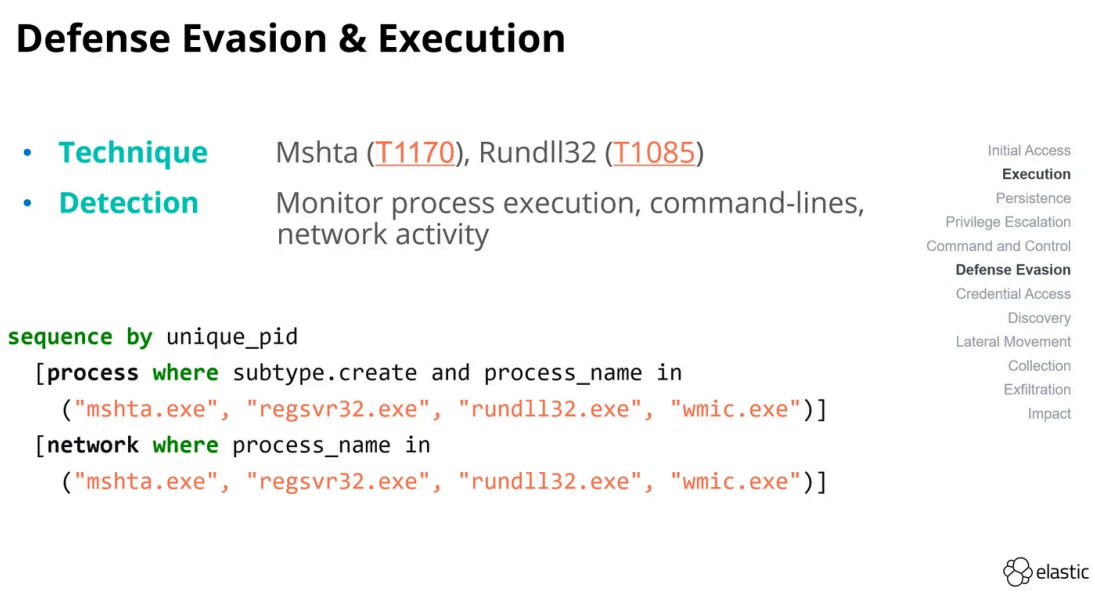
Defense evasion/execution

Mshta (T1170), Rundll32 (T1085)

Tools like Koadic often include some usability features that help facilitate payload building, also known as stagers. These small pieces of code get executed on the victim machine and are used to establish a network connection back to the attacker in order to bring in a staged payload for execution. Stagers represent a significant portion of the early phases of the intrusion process. The following example continues exploring the detection strategy for a variety of Windows utilities used to proxy execution with Koadic stagers.

The EQL query below uses the sequence operator, a function of EQL that matches based on the order of events in a sequence. In this case, we are matching when one of these standard Windows administration utilities initiates a network connection. Where another language might require an analyst to write several rules — one for each of these utilities — EQL enables us to build an array capable of matching many permutations.

Using the filter operator joins these events in sequence by their process identifier (PID). I like this example because it's capable of detecting malware and other offensive tools that aren't specific to Koadic. With that said, it might take a little filtering to remove potentially benign events such as network administrative activity, but this kind of behavior is something every organization should be tracking and reviewing on a certain cadence.



Defense evasion and execution - Mshta

Defense evasion & execution - stagers

EQL query:

```
 sequence by unique_pid
[process where subtype.create and process_name in
("mshta.exe", "regsvr32.exe", "rundll32.exe", "wmic.exe")]
[network where process_name in
("mshta.exe", "regsvr32.exe", "rundll32.exe", "wmic.exe")]
```

One of the more interesting takeaways when reviewing offensive tooling is finding the different artifacts that get left behind unintentionally. All it takes is one “loud” artifact, such as a file or registry modification that sticks out, to quickly find suspicious activity.

In Koadic’s case, HTTP stagers use Internet Explorer’s core architecture to make a web request by default, which causes the stager to be created within the Temporary Internet Files directory. This behavior occurs due to the way Internet Explorer caches browser content to quickly load web pages. Monitoring this kind of behavior with certain executables can lead to reliable detections outside Koadic, such as generic cradles used to download and execute malicious code.

Defense evasion & execution - cached stager

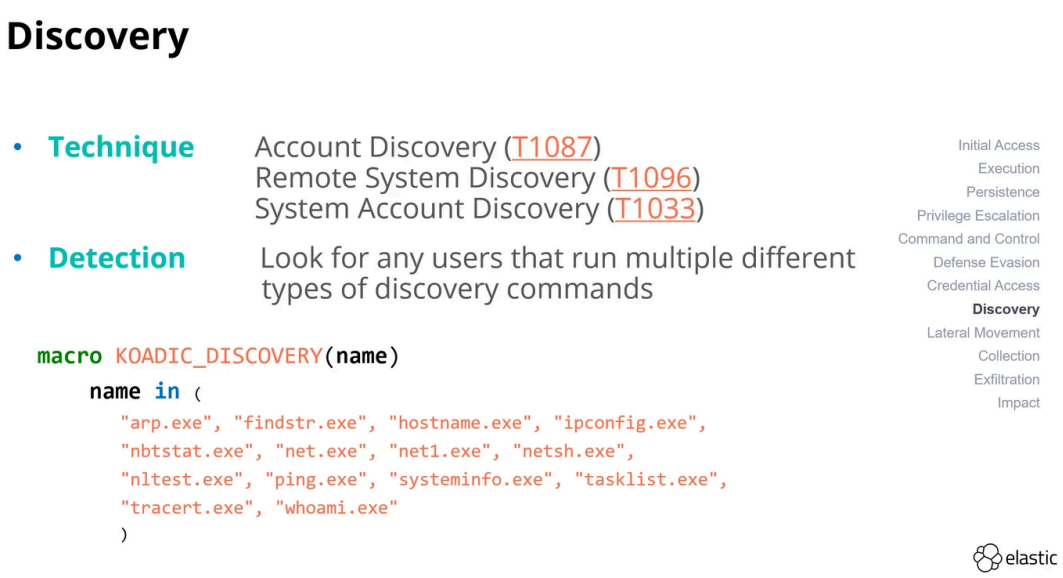
Defense evasion & execution - cached stager

EQL query:

```
 file where process_name in
("mshta.exe","regsvr32.exe", "rundll32.exe", "wmic.exe")
and subtype.create and file_path == "*Content.IE5*"
```

Discovery

Account Discovery (T1087), Remote System Discovery (T1096)



Discovery - macro

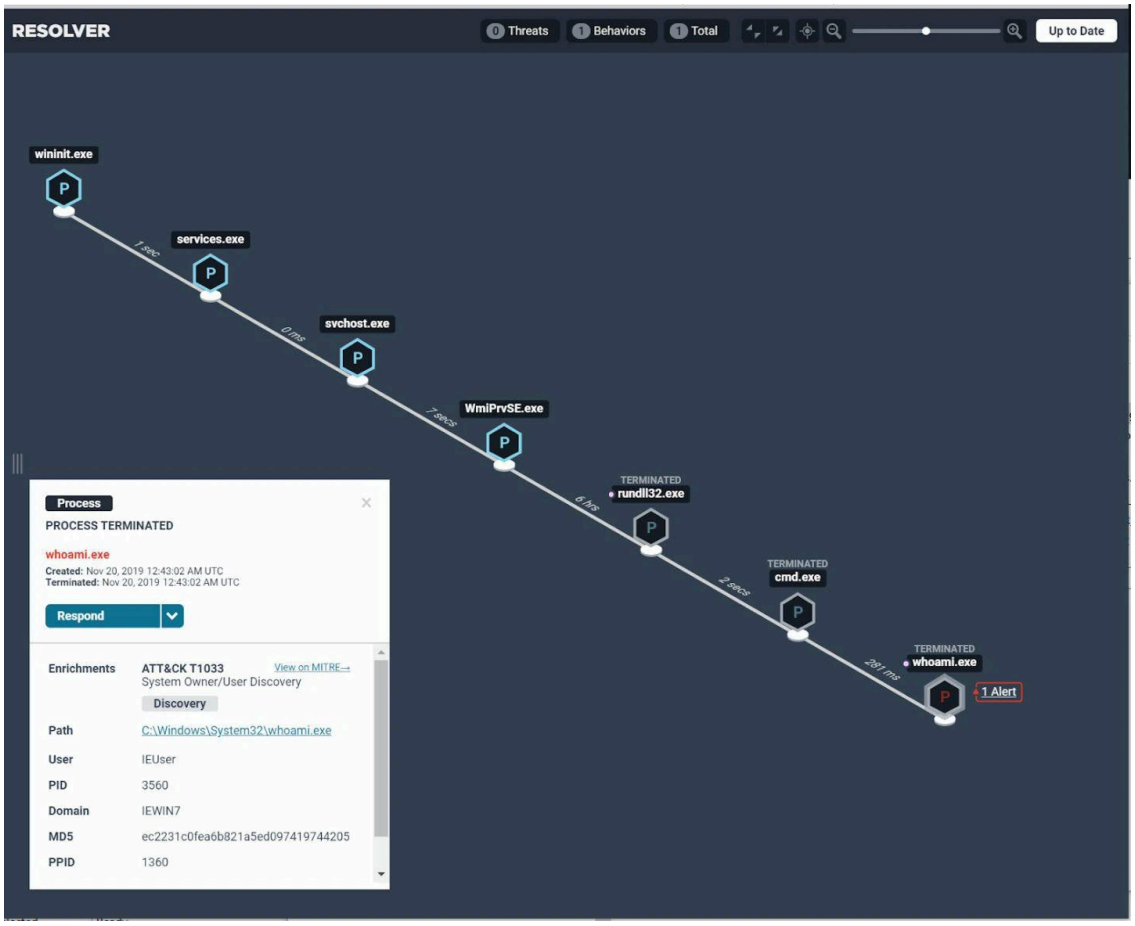
Discovery - macro

A feature of EQL is the ability to share or reuse similar logic between queries. By using macro declaration, we can bundle a collection of items together and call the array like a variable. A good example would be grouping Microsoft Office applications into a macro, or, in this case, several different Windows programs that can be used for discovery and enumeration.

EQL query (macro):

```
 macro KOADIC_DISCOVERY(name)
name in (
"arp.exe", "findstr.exe", "hostname.exe", "ipconfig.exe",
"nbtstat.exe", "net.exe", "net1.exe", "netsh.exe",
"nltest.exe", "ping.exe", "systeminfo.exe", "tasklist.exe",
"tracert.exe", "whoami.exe"
)
```

The Elastic Endpoint Resolver view below helps provide some context about how Koadic spawns child processes. By using the Koadic module (exec_cmd), and running a natively supported command such as “whoami /groups”, we can see the Rundll32.exe application was invoked by WmiPrvse.exe and passes instructions down to the command prompt before launching the Whoami.exe application.



Elastic Endpoint Resolver

Elastic Endpoint Resolver visualization

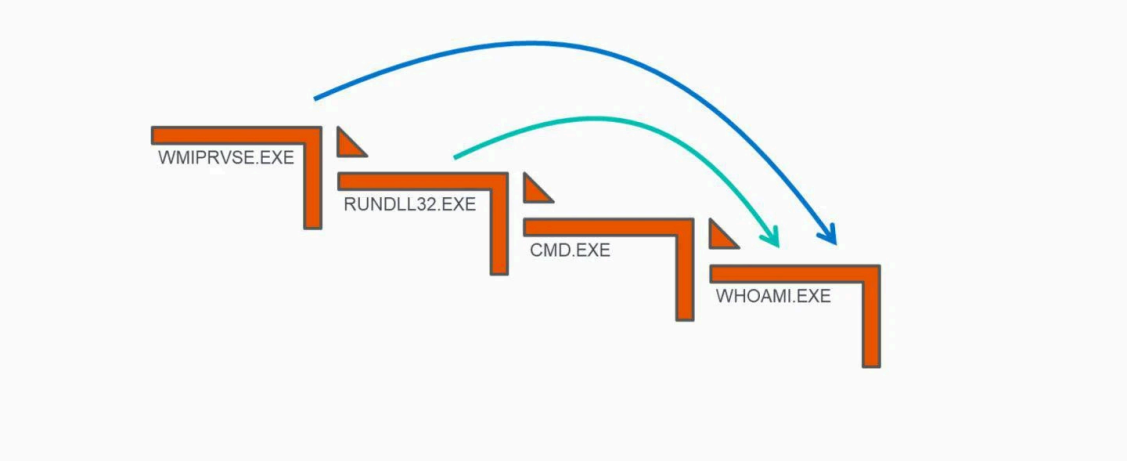
Now that we have a better understanding of the attack chain, let's tie our previous macro (KOADIC_DISCOVERY) into a sequence-based detection looking for three process creation events from any one of those enumeration programs within a period of 10 minutes, executed by the same user. This same feature would let you monitor for processes that were previously observed. For example, maybe 15 days later the process makes a network connection to pull down an additional payload. What other language lets you find long-term payloads that sit dormant for weeks or months?

EQL query:

```
 sequence by user_name with maxspan=10m
[process where subtype.create and KOADIC_DISCOVERY(process_name)]
[process where subtype.create and KOADIC_DISCOVERY(process_name)]
[process where subtype.create and KOADIC_DISCOVERY(process_name)]
| unique user_name
```

The query above is fully-functional and can be used as a generic detection for initial discovery and enumeration. But what if we had some reason to tighten the logic around Koadic specifically? Understanding the process genealogy of Koadic at the endpoint level, we can leverage different process relationship tracking functions in EQL such as **child of** and **descendant of**.

By using the **child of** function and setting the parent process to Rundll32.exe, we are essentially getting the grandchildren of Rundll32.exe. Then if we wanted to take it even further, we can add the **descendant of** parent process WmiPrvse.exe. This example demonstrates the flexibility of EQL to provide powerful detection capabilities for real adversary behavior.



Process tree visualization - child of and descendant of

Process tree visualization - child of and descendant of

EQL query:

```
 sequence by user_name with maxspan=10m
[process where child of [process where parent_process_name == "rundll32.exe"]
and KOADIC_DISCOVERY(process_name) and
descendant of [process where parent_process_name == "wmiprvse.exe"]]
[process where child of [process where parent_process_name == "rundll32.exe"]
and KOADIC_DISCOVERY(process_name) and
descendant of [process where parent_process_name == "wmiprvse.exe"]]
| unique user_name
```

Privilege escalation

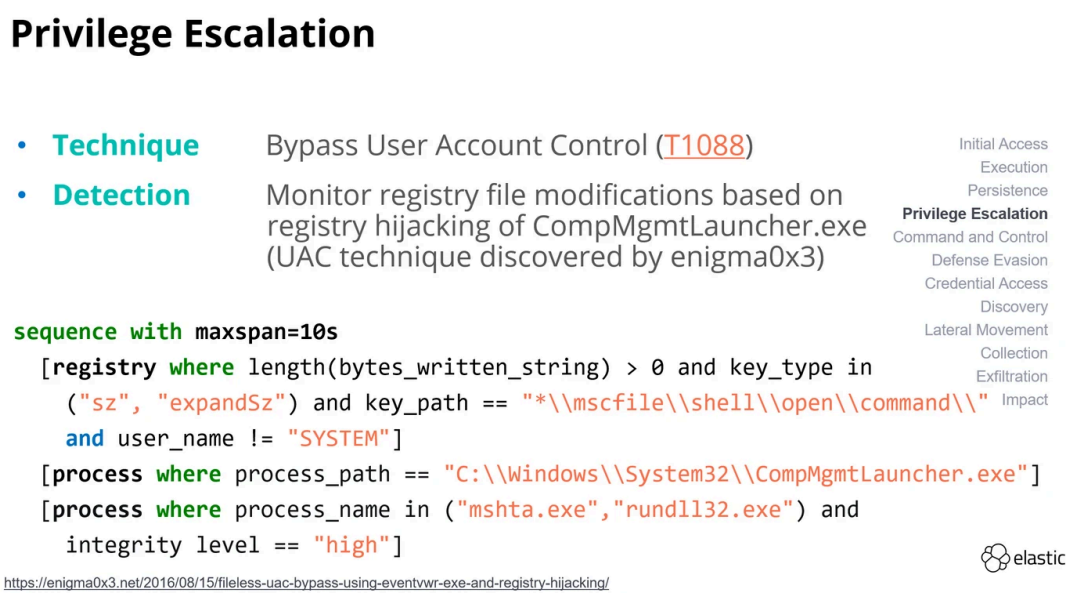
Bypass User Account Account (T1088)

While attackers control targeting of victims in many cases, they don't always wind up with an elevated user during initial compromise. Even when a spearphishing victim is a local administrator, the attacker will oftentimes need to escalate from a Medium to High integrity process before continuing. Off-the-shelf offensive tools like Koadic can enable that transition with relative ease, including several different UAC Bypass modules out of the box.

For this example, we'll examine a well-known UAC Bypass technique published by Matt Nelson (@enigma0x3) while leveraging the Computer Management launcher — CompMgmtLauncher.exe — which is interoperable with the Microsoft Management Console (MMC). Details about this technique, which still works on Windows 7 endpoints, can be found here.

This technique involves modifying the Windows Registry to change the default association of files the MMC interacts with (HKCU\Software\Classes\mscfile\shell\open\command) to an application of the attacker's choice. By deploying a malicious script object with a compatible extension and altering this registry key value to launch a built-in script interpreter, an adversary is able to circumvent controls.

Right after this registry modification, look for the new process creation event tied to the auto-elevated Microsoft program (CompMgmtLauncher.exe), followed by common Koadic stager descendant processes such as Mshta.exe or Rundll32.exe — processes that should be running in a high integrity context. We can combine those events into an ordered sequence and constrain the total run-time for all the steps to complete within 10 seconds.



Privilege escalation - UAC bypass

Privilege escalation - UAC bypass

EQL query:

```
 sequence with maxspan=10s
[registry where length(bytes_written_string) \> 0 and key_type in
("sz", "expandSz") and key_path == "*\\mscfile\\shell\\open\\command\\"
and user_name != "SYSTEM"]
[process where process_path == "C:\\Windows\\System32\\CompMgmtLauncher.exe"]
[process where process_name in ("mshta.exe","rundll32.exe") and
integrity_level == "high"]
```

Collection/exfiltration

Data from Local System (T1005)

Koadic's method of C2 may be interesting to analysts of several kinds due to the transactional way it exchanges data between implants and server. This behavior is highlighted through some direct examples of specific

commands executed below:

```
cmd.exe /q /c chcp 437 & time 1> C:\Users\IEUser\AppData\Local\Temp\95fe63d2-e79d-2706-2e89-2084a225343e.txt 2>&1
cmd.exe /q /c chcp 437 & hostname 1> C:\Users\IEUser\AppData\Local\Temp\9909f618-4fb5-eb66-745d-f40143687330.txt 2>&1
```

Specific commands

Command shell redirection into text files

Koadic redirects STDOUT/STDERR to a temporary text file that stores the output of the operator's commands as they were presented to the server. These commands are then read back into the Koadic C2 terminal. One second after this file is initially created, it is automatically deleted.

With the right endpoint visibility, malicious behaviors you might be incapable of otherwise detecting stand out. To demonstrate a detection around this, we will use the event of function to filter only for processes that come from cmd.exe that contain a redirector (\>), then tie the PID of that process to same PID that performed file activity related to the text (.txt) file activity.

EQL query:

```
 file where file_name == "*.txt" and
event of [process where process_name == "cmd.exe" and command_line == "*\>*"]
```

Subtype	Filepath
File Created	"C:\Users\IEUser\AppData\Local\Temp\95fe63d2-e79d-2706-2e89-2084a225343e.txt"
File Created	"C:\Users\IEUser\AppData\Local\Temp\9909f618-4fb5-eb66-745d-f40143687330.txt"

Example results

Example results showing file modification

If you wanted to get more context, such as what command was passed from Koadic, we can turn the detection into a sequence and add the process event.

EQL query:

```
 sequence with maxspan=5s by unique_pid
[process where subtype.create and process_name == "cmd.exe" and command_line == "*\>*" and
descendant of [process where process_name == "wmiprvse.exe"]]
[file where subtype.create and wildcard(file_name, "*.txt", "*.log")]
```

Event Type	Command Line / Filepath
Process	"C:\Windows\system32\cmd.exe" /q /c chcp 437 & whoami /groups 1> C:\Users\IEUser\AppData\Local\Temp\2a0f4991-b684-afe0-63e6-207e58ac4af8.txt 2>&1
File	"C:\Users\IEUser\AppData\Local\Temp\2a0f4991-b684-afe0-63e6-207e58ac4af8.txt"

Example results

Example results combining process and file modification

Conclusion

To summarize, we analyzed one offensive framework (Koadic) and several of its prominent features, reviewed a flexible query language (EQL) to express detection logic, and stepped through several ways to identify behavior tied to Koadic with example analytics.

I see the availability of offensive frameworks like Koadic as an opportunity for defenders, and a motivation to stay vigilant. Thinking creatively about how to detect these post-exploitation behaviors and assessing how these tools perform against their own detection capabilities will put an organization on a path to greater success in stopping similar threats.

To help enable organizations, we've added all the queries in this post into the EQLLib repository. For readers interested in the original presentation from BSidesDFW this year, here is a link to the slides.

EQL support is being added to Elasticsearch.
