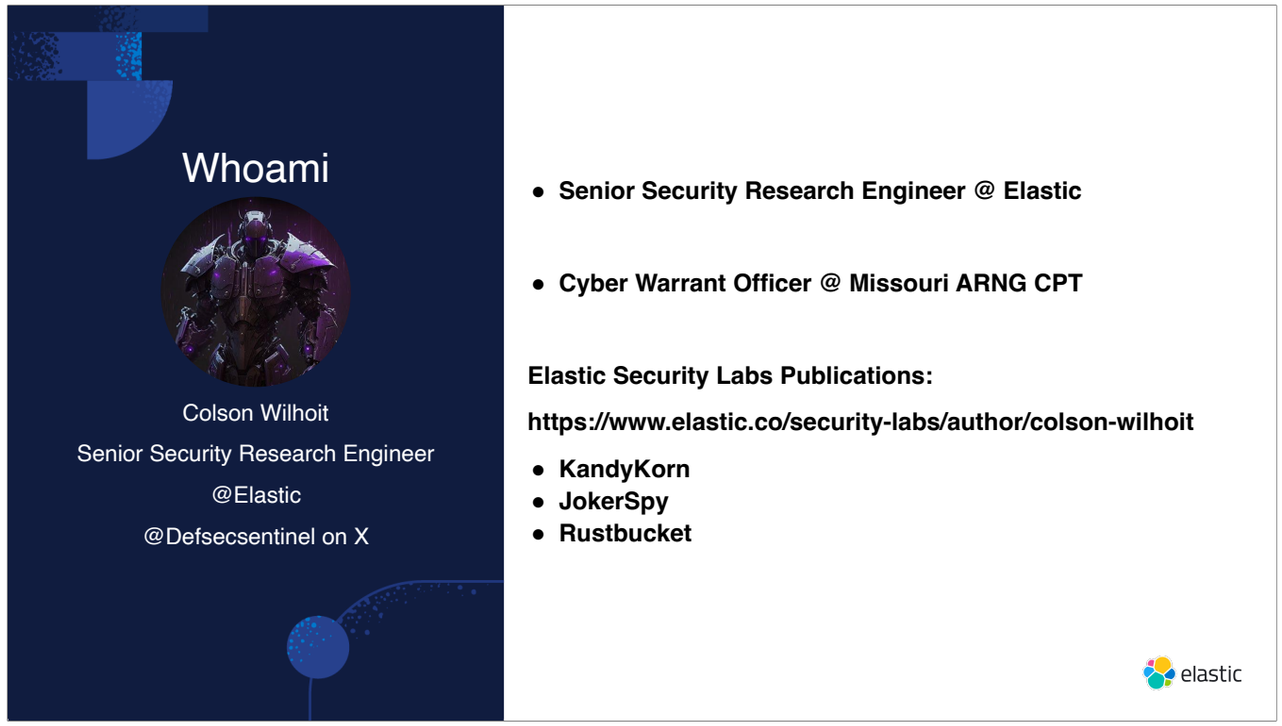
Aloha Everyone, welcome back from lunch. It’s great to be here.

I’m excited to get to present TripWires in the Dark Detecting and Preventing sophisticated threat actors and previously “undetected” malware on macOS with Behavior Detections



A little bit about me before we start. My name is Colson Wilhoit but many of you may know me as Defsecsentinel on X.

I work at Elastic on the Threat Research and Detection Engineering team as a Senior Security Research Engineer where I am responsible for threat research developing signatures and behavior detections for both the endpoint agent and SIEM.

My primary focus is macOS where I also drive the direction and feature development of the Elastic Agent and Elastic Defend for macOS

You may also know me from my write-ups surfacing threats like KandyKorn, JokerSpy and RustBucket

Aside from that I’ve served coming up on 10 years in US Army National Guard almost the entirety of that in the Cyber Branch

Agenda:

My hope today is that I can paint a picture and take you on a journey that ends in your understanding of how important and powerful Behavior detections are on macOS.

We will seek to understand at a very high level the current threat and defense landscapes

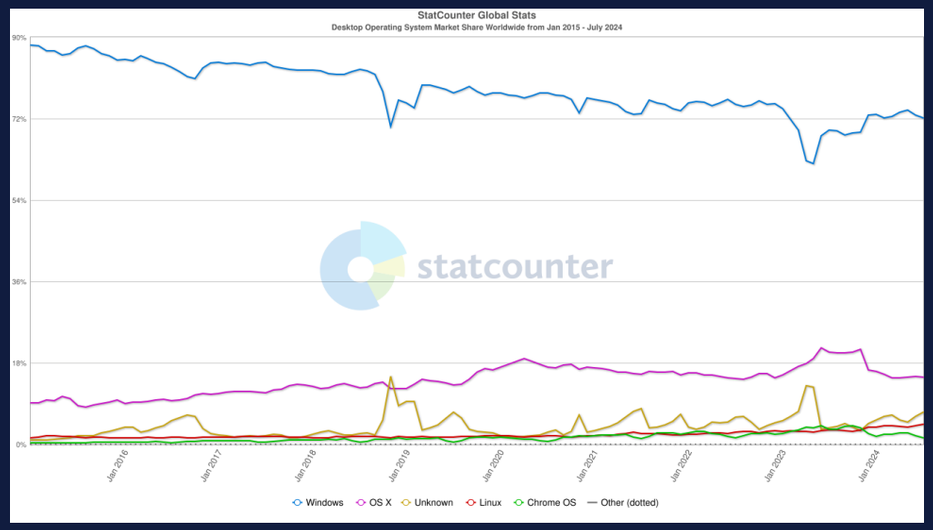
We will take a very brief high level look at limitations and gaps in the defense landscape and how that points to Behavior detections as the possible solution

We will do some hands on detection engineering of Endpoint Behavior Rules

And then Finish up with some real world Case Studies, where we are heading with Detection and Key Takeaways

Let's start by examining the macOS threat landscape at a high level as it stands today.

MacOS Adoption Growth



MacOS Desktop Market Share Increase Year over Year

elastic

MacOS adoption continues to increase as you can see here via StatCounter showing macOS with the Purple color

When we look at the core verticals within the enterprises that are using macOS heavily this is what we see.

MacOS Adoption Core Verticals

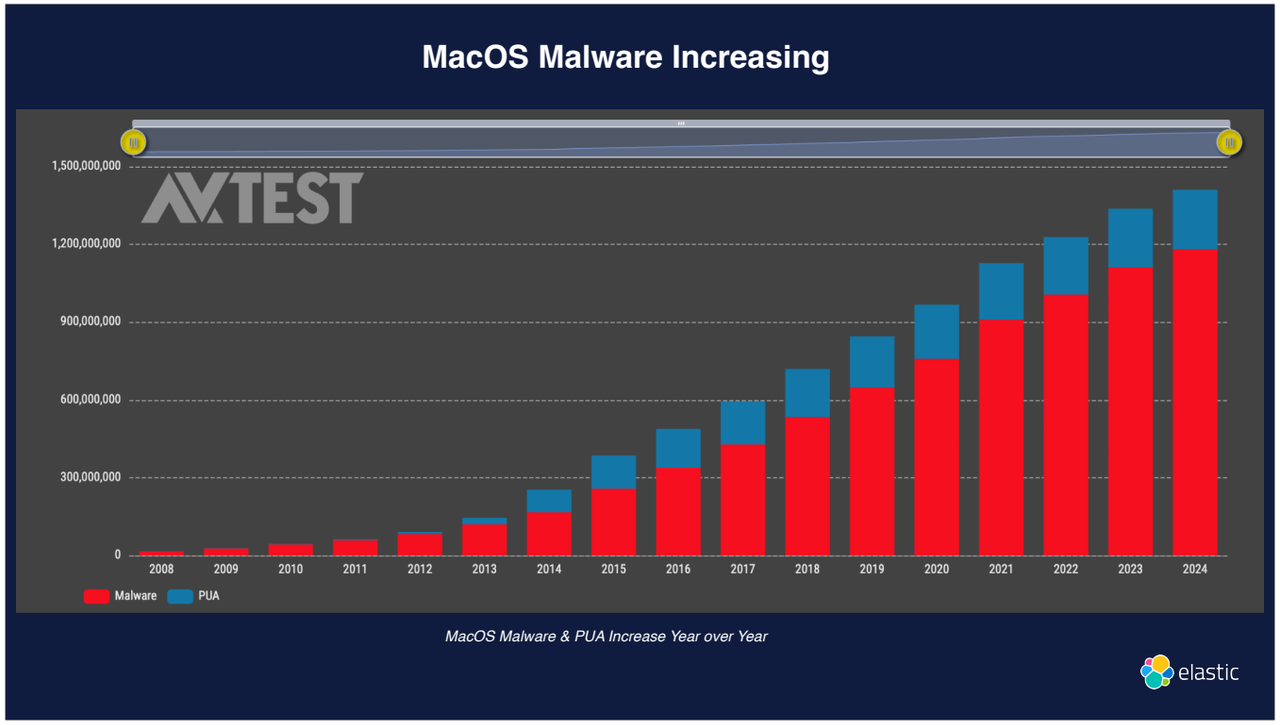
- Finance
- Software Development
- Design and Creative Industries
- Education
- Enterprise and Corporate Environments
- Media and Entertainment

elastic

MacOS isn’t just used by and for the creatives anymore.

Adoption has been strong in sensitive verticals where users have access to sensitive systems and data.

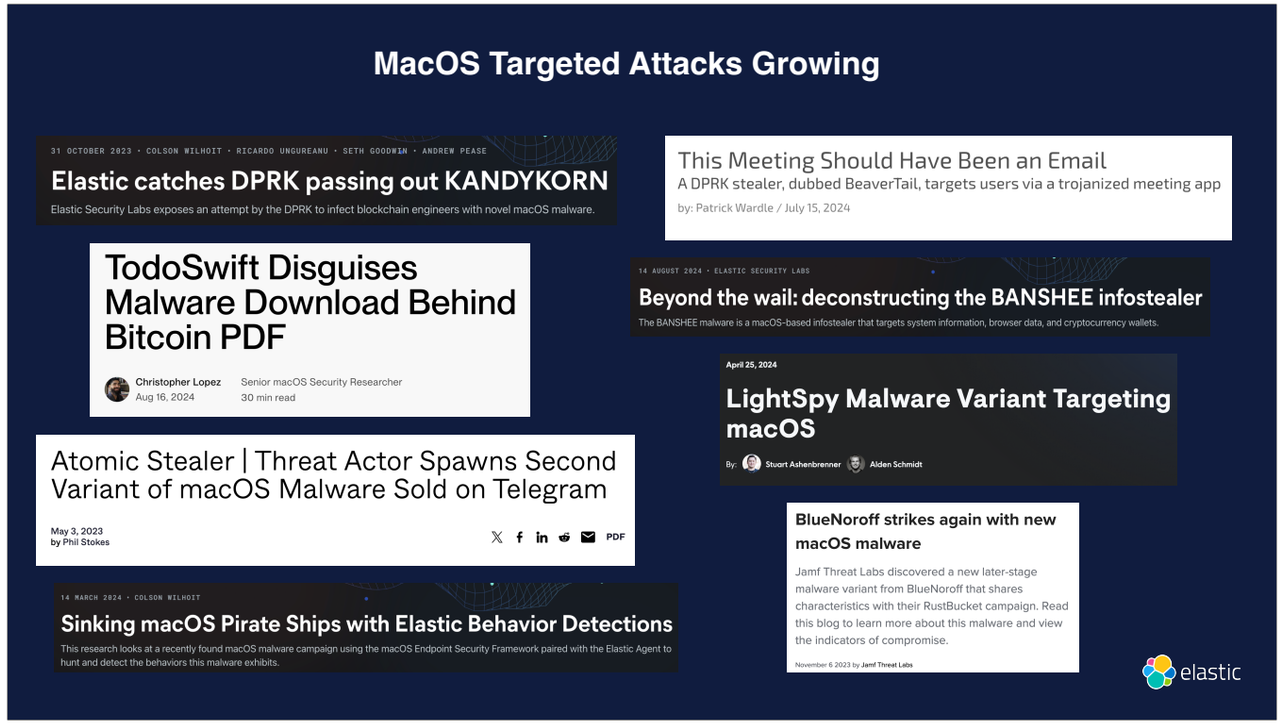
In verticals like Software Development, Finance, and Enterprise Users (to include many C-Level individuals).



MacOS Malware & PUA Increase Year over Year

Of course with an increase in macOS adoption across the enterprise and in sensitive industry verticals you would expect to see threat actors turn their attention to macOS as a target providing valuable access to the users and data they want.

AVTest here shows macOS malware and unwanted applications increasing dramatically year over year.

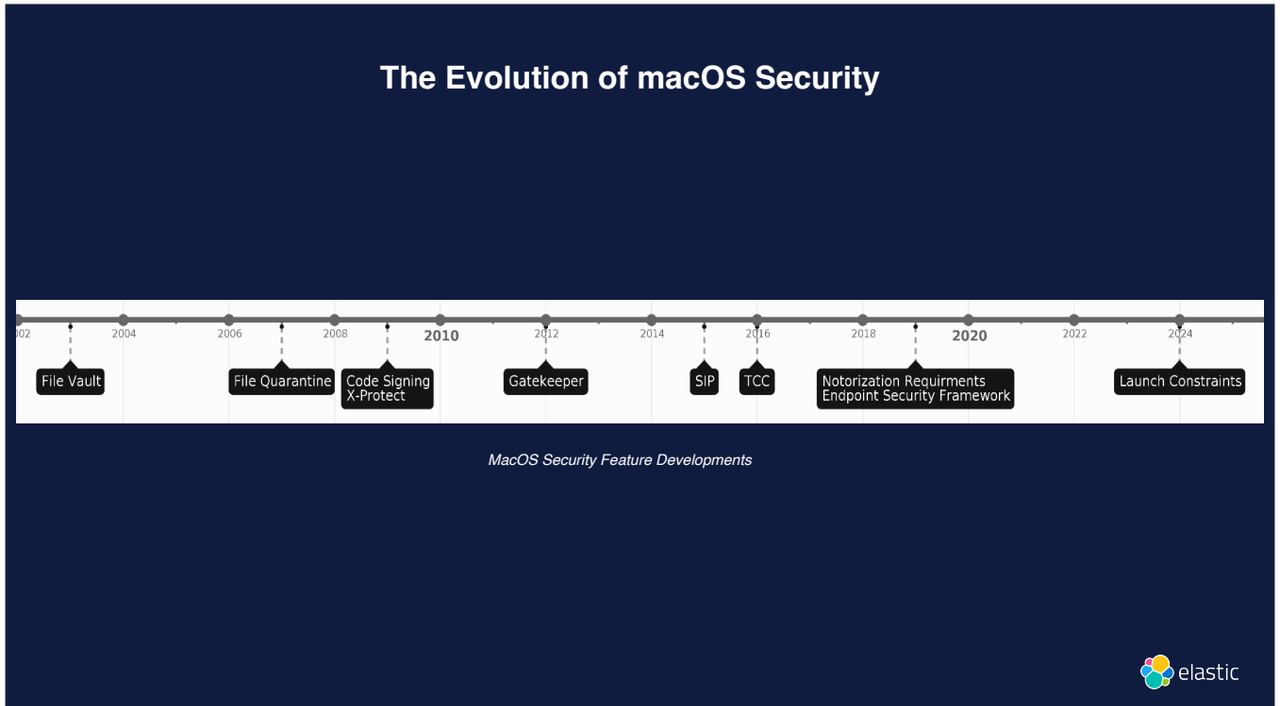


Campaigns and payloads purpose built, targeting macOS users all over the world are increasing each year from informations stealers to advanced, multi-stage nation state intrusions.

Lots of money and development time is being spent to compromise mac systems.

This re-enforces the need for innovation as defenders of macOS and IOS along with the responsibility to be vigilant and really shape the threat landscape.

I think its important to ensure we are on the same page when it comes to what our Defense Landscape for macOS looks like right now along with the limitations and gaps that exist as well



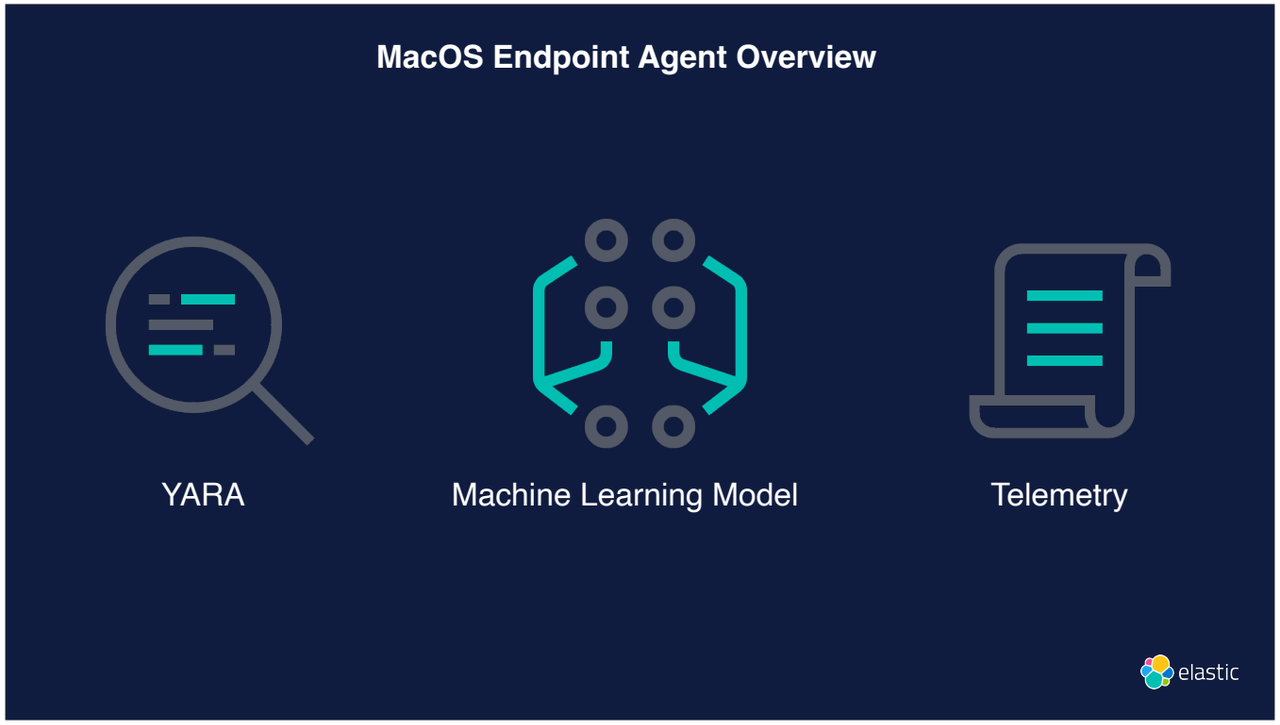
MacOS Security Feature Developments

MacOS security has improved quite dramatically over the years since its inception starting with features like Filevault and continuing with Gatekeeper, SIP, and TCC all the way up to Launch Constraints recently

Apple has shown a strong propensity for protecting user data and implementing common sense security controls that go a long way in providing a strong framework against many threats

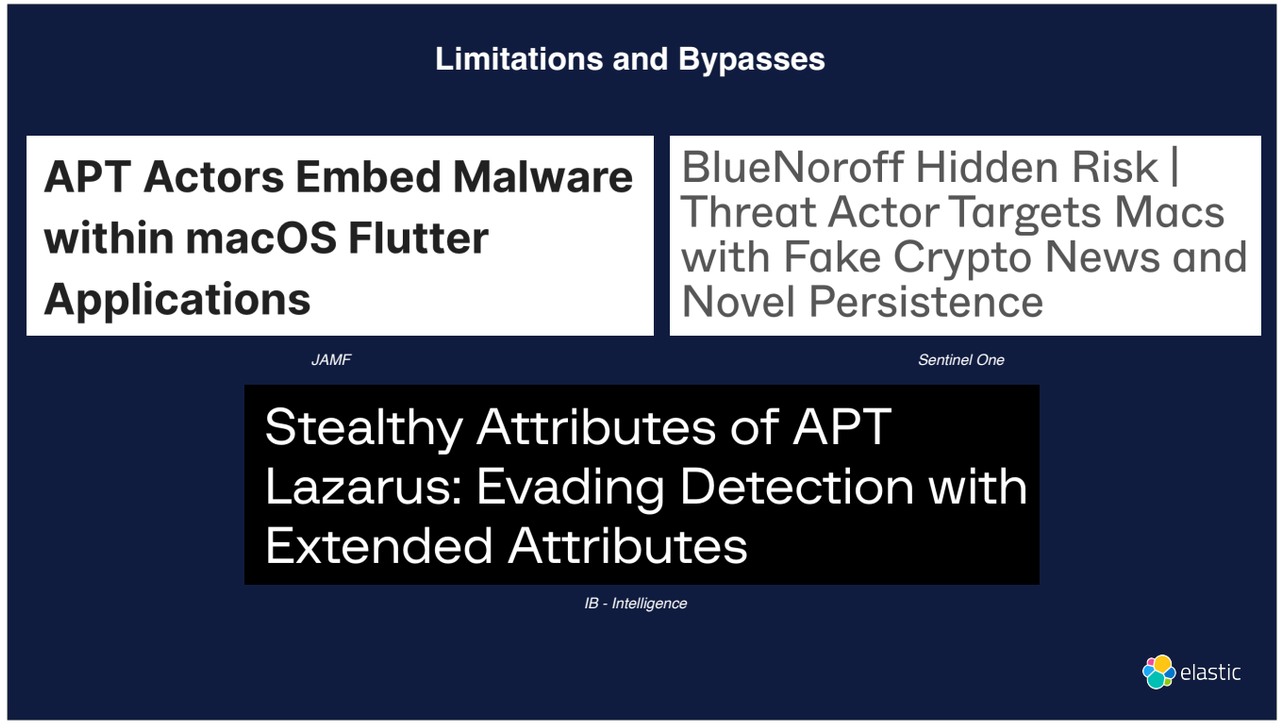
Also incredibly important, as we will see and reinforce later on, is mac system telemetry data.

MacOS provides the Endpoint Security API now which is a security event subscription service for 3rd party software to utilize as well as macOS unified logs which encompass data sources from all over the os to include application logs, additional security logs and more.



We need to at least mention MacOS EDRs, specifically their endpoint agents, and how they work at a high level if we are going to understand the complete defense landscape as they are becoming more common and prominent within enterprises today.

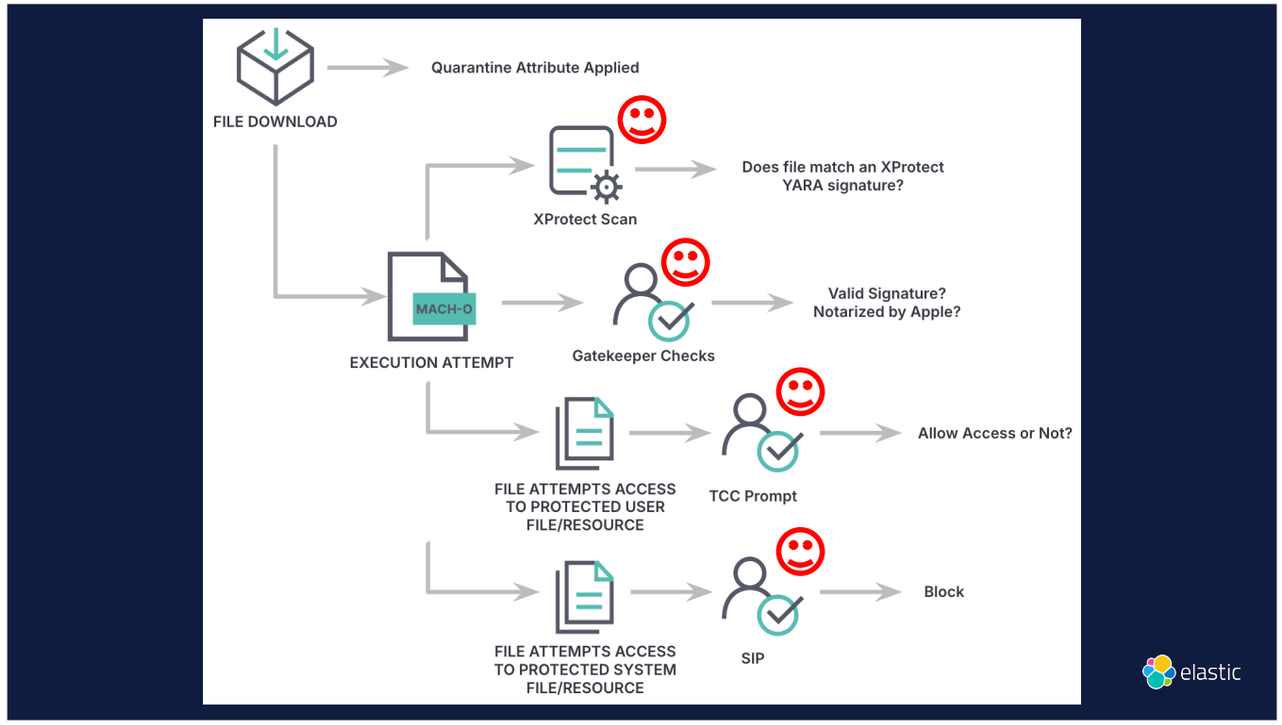
MacOS EDRs can be divided up into 3 core functions essentially. File and Memory focused YARA signatures, Mach-O Machine Learning Models, and Telemetry, which today usually encompasses ES and Unified Logs. All of this allows the EDR to take action on these and feeds back to a SIEM or control console of some sort.



Of course all of these wonderful macOS security controls and features along with the core defense mechanisms EDRs on macOS have limitations, shortcomings and bypasses.

These headlines you see here are just 3 very recent write-ups by Jamf, SentinelOne, and IB that surfaced malware, written in different languages (Rust, Golang, and Flutter) that while active where signed and notarized by Apple with 0 VirusTotal hits, no or almost no YARA signatures, and bypassed most ML models.

This is a lot more common than you think. Threat actors find creative ways to get code execution



Bypassing YARA is not extremely difficult as it detects what has been known, you can test against it, it requires a pretty stringent development pipeline to stay on top of and generally, at least right now, isn't effective against script based payloads

In most cases if your payload is signed and notarized you will sail right through Gatekeepers checks or if you are using a Python application or Nodejs application, like we see with many of the initial access lures today, bypass it altogether and if you take advantage of tools like Curl or Nscurl to download follow on payloads those files are not subject to Gatekeepers checks

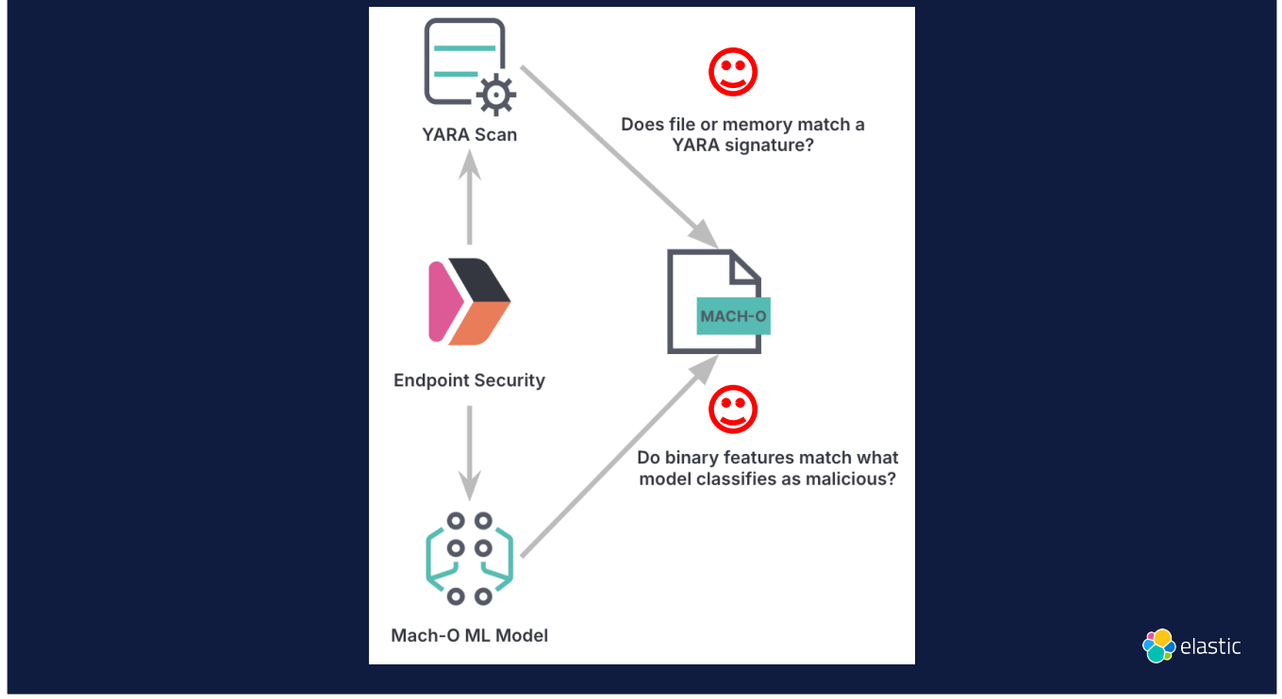
Regarding TCC you can spawn within the context of something like Terminal or another app that may already have those permissions and access granted to them allowing us to inherit those, which we commonly see especially with InfoStealers, or the user may just grant the request for a myriad of reasons like prompt fatigue, desire to use the application, lack of education etc.

For SIP just avoid those protected system/file resources

Of course all of these controls also have vulnerabilities that are uncovered all the time and exploits for past macOS versions

But you are in the door with many of these common methods of bypass, we've seen, that can cascade down to the EDRs endpoint agents as well.

Not to say all of these controls and efforts are not critically important, which they are, especially layered together but you can start to see the threat actor sized hole



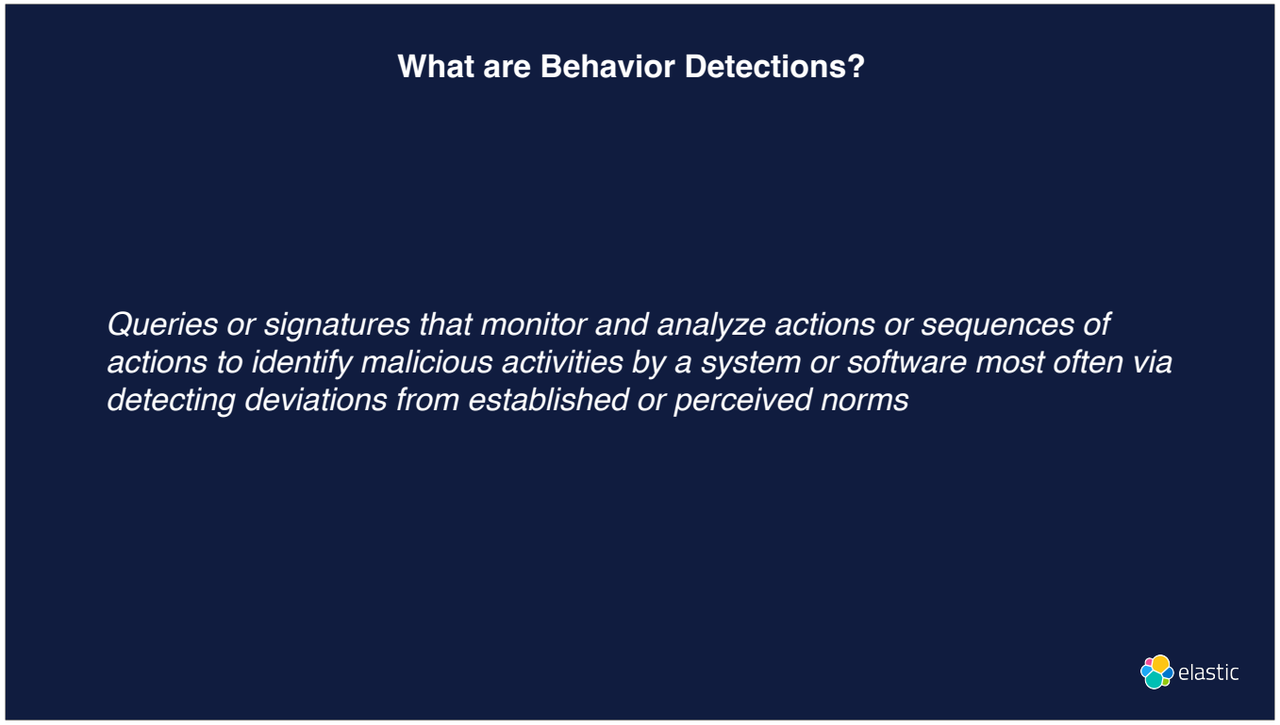
Not always but with the samples we just mentioned odds are that if a binary or application gets signed and notarized its not triggering Apple's XProtect signatures and won't trigger EDR YARA signatures either especially if it's a fresh, new, custom built payload and more likely if its nation state developed.

In regards to the ML models they can be powerful but in the end are only as good as the data they are trained on and the training pipeline they use to include the samples it's fed which must be continually up-kept

Techniques like packing, obfuscation or the "Extended Attribute Smuggling" IB found recently go a long way in avoiding ML classification as well since the models rely on specific feature extraction

Defense in depth is incredibly important and makes threat actors objectives more difficult to obtain but gaps exist and I believe we should look towards Endpoint Behavior detections as one strong solution to bridge that gap

So let's delve into Behavior Detections.



What are Behavior Detections?

Behavior Detections are : READ DEFINITION

Breaking that down real quick

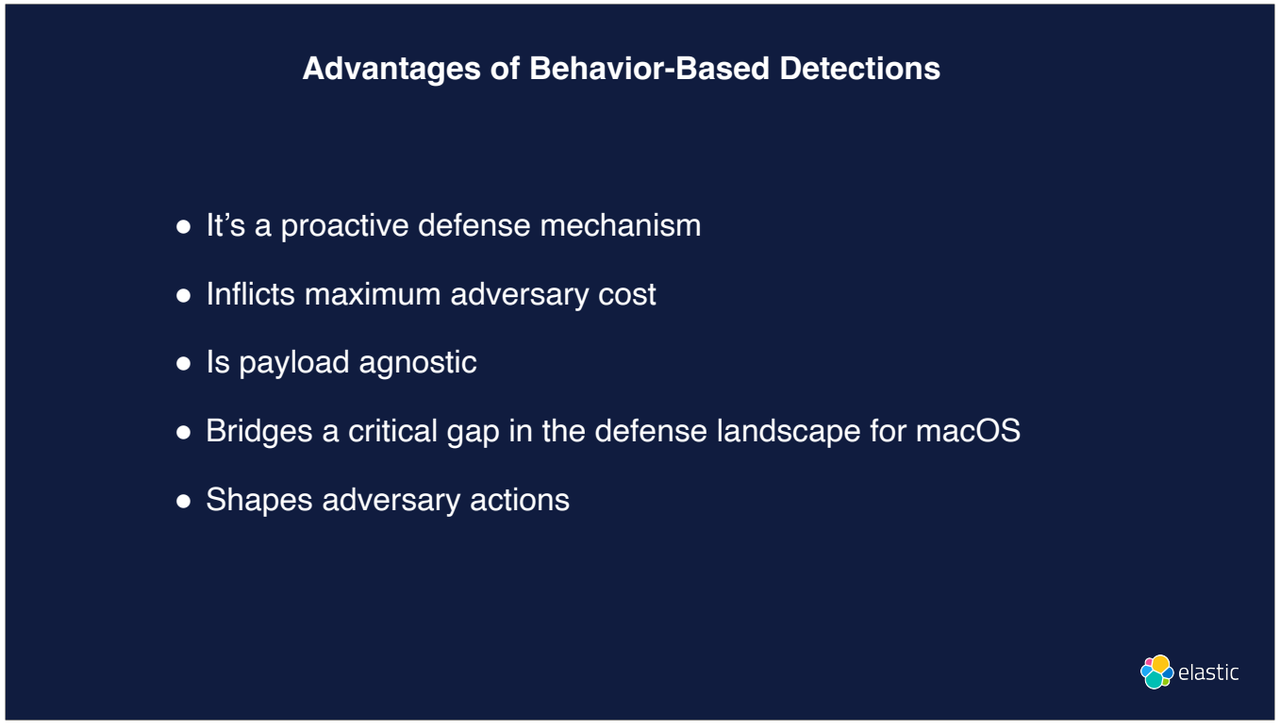
Queries or Signatures, this is referring to the language you use to query system telemetry data, Endpoint Security events.

That monitor and analyze actions or sequences of actions to id malicious activities by a system or software

Every meaningful action has consequences or produces artifacts and being able to sequence chains of actions or events together to create a fingerprint of that behavior is powerful as we will see

Most often via detecting deviations from established or perceived norms

Many if not most of the behaviors threats and their tools exhibit intentionally or unintentionally usually deviate from normal user or Apple prescribed behavior, especially on macOS.



So what are the advantages of implementing robust, resilient Endpoint or SIEM Behavior Detections?

It’s a Proactive defense mechanism. Previous Behavior rules can detect future or new Malware samples

They inflict the maximum adversary cost forcing them to expend the most time, energy and resources in order to evade. Go check out Dave Bianco's Pyramid of Pain if you want a better understanding of this.

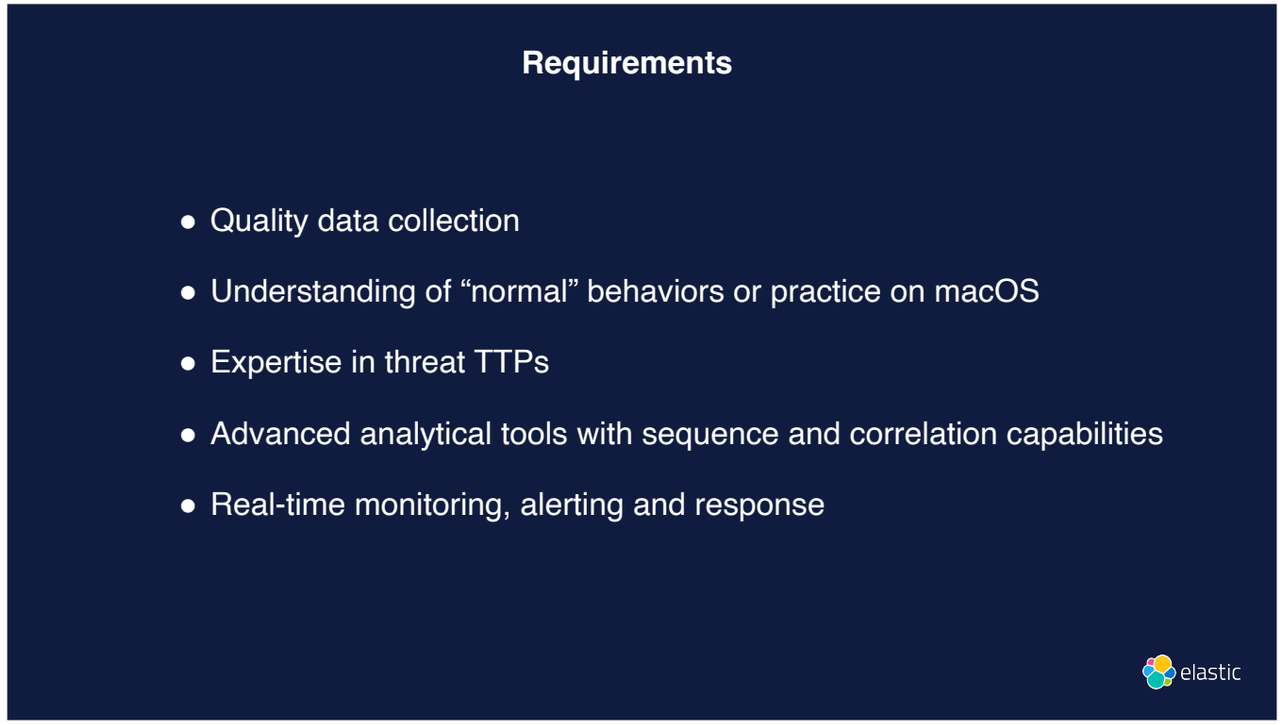
These rules are payload agnostic. Regardless of the language or type of payload, if they take action on the system, our Behavior rules can detect and/or prevent them

They bridge the threat actor sized gap that exists between YARA, ML and macOS Security Controls that we have seen malware samples slip into time and time again

Finally they allow us to shape adversary actions.

If a threat actor gets caught or even sees and analyzes our Behavior rules and adapts we have just shaped their options and forced them to do something different.

And usually when attackers attempt to circumvent or evade detection they actually end up generating new behaviors that can be detected.



There are of course requirements in order to effectively put into practice Behavior Detections that will impact the threat landscape. Hopefully some of these are obvious.

First and foremost is Quality Data Collection.

That doesn’t mean ingesting every ES event under the sun and on the opposite end not just ingesting a small subset of basic ES events.

You actually don’t need every ES event to detect and prevent 99% of threats targeting macOS believe it or not

You also need to enrich the ES events you do ingest in order to properly sequence and correlate the data.

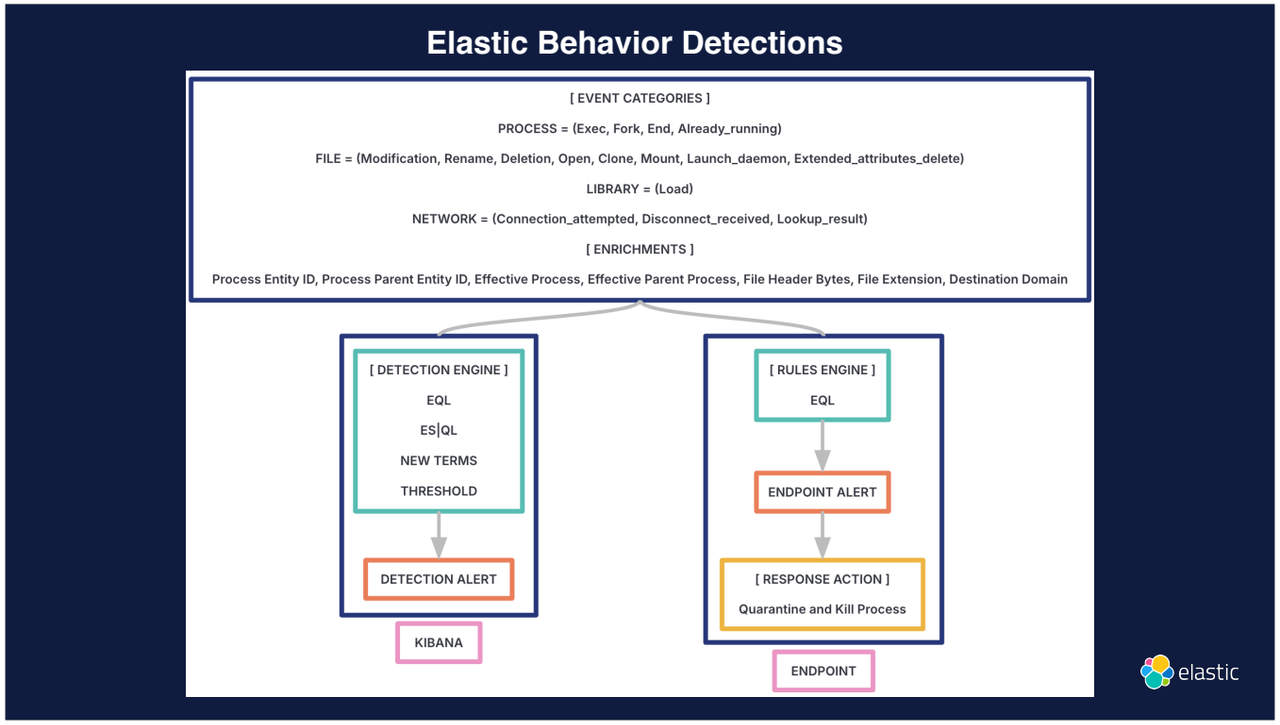
In order to collect quality data and create Behavior Detections you need to understand “Normal”.

Like we mentioned earlier Apple has designed an OS in macOS that is meant to behave in a very specific way and in regards to how both users and developers use it. Understanding that and how things work and look will enable you to identify outlying behavior

You need to study and understand threat actors targeting macOS. What are they after? What do they have to do to get there?

Advanced Analytical tools are a must. If you can’t take singular events (like a process execution) and sequence it or correlate it to a file or network event in a way that makes sense you are not going to be able to build Behavior rules that are strong

And finally you need to be able to monitor the events you collect in real time, alerting (Behavior Rules) and responding (quarantining files and killing process tree)



Here is a visual breakdown showing how we at Elastic implement Behavior Detections on both our Endpoint and SIEM obviously focusing on macOS here

On the endpoint at the data level we organize or group the events we collect, both ES and custom, into general event categories, process, file, library and network.

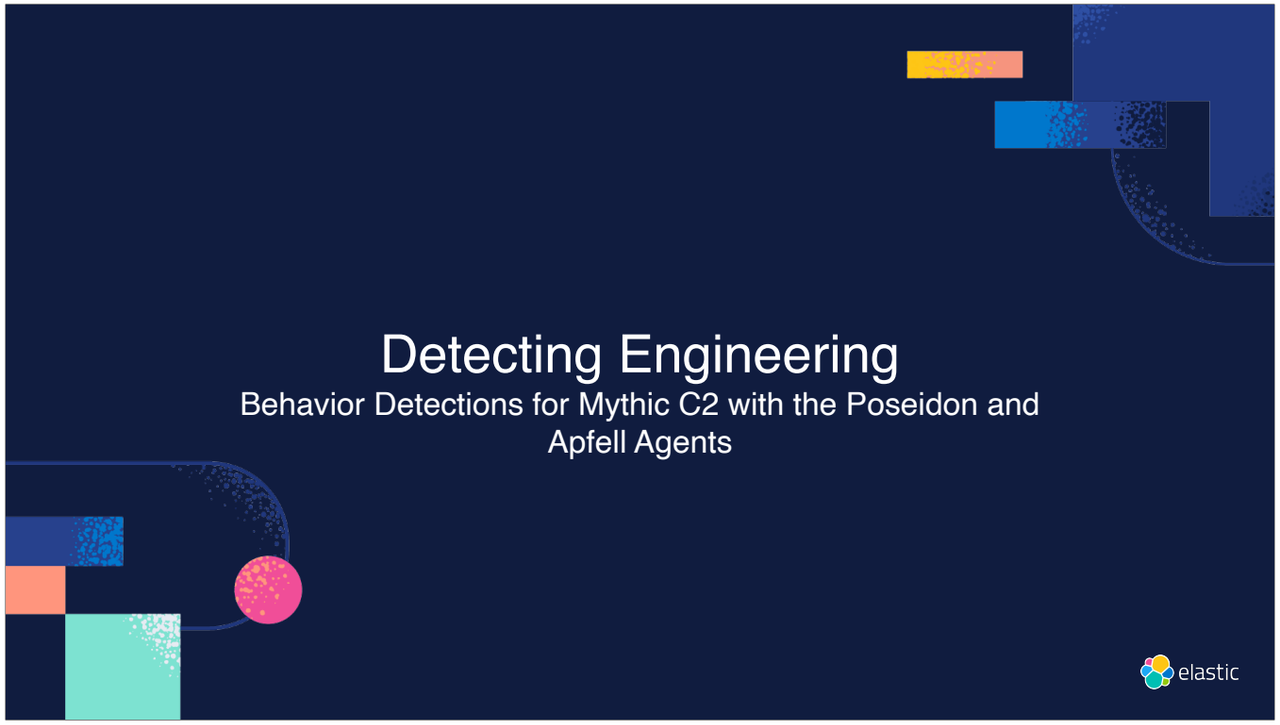
You can see some of the unique events we have created and collect like Launch Service plist collection, Dylib load events, and soon DNS events.

We then enrich those events with things like a unique process tree identifier we call the entity id, effective or responsible process and parent process data, process code signature data, dylib code signature data, process environment variables, executable file header bytes, destination domain and more.

Those events are then simultaneously sent to the Elasticsearch database and fed through our Endpoint Behaviour Rules Engine. On the Endpoint when the events go through our Rules Engine, in real time, they are evaluated against my Endpoint Behavior rules that are built using Elastic's Event Query Language.

If an event or sequence of events match a Rule an Alert is generated, and if applicable the file is quarantined and the process along with its tree is killed.

On the SIEM side in Kibana, Kibana visualizes and queries the data stored in Elasticsearch FYI, we have our Detection Rules Engine, which allows users to create Behavior detection rules using any of our query languages and take advantage of some powerful rule types like new terms (first time a behavior or event has been seen) or threshold (if an event occurs a certain amount of times) to trigger alerts based off the data available in Elasticsearch.



We are going to do some Behavior Rule Detection Engineering taking a look at a couple of interesting and advanced techniques found available within the Poseidon and Apfell agents with the awesome Mythic C2 framework.

And up until now I'm pretty sure 99% of people to include the authors didn't think these techniques could be fingerprinted using Behavior rules.

We are going to breakdown these techniques and show how you can, with quality data and the right tools create solid detections for the Behaviors these techniques exhibit

Mythic C2: Poseidon and Apfell Agents

- Cross-platform containerized C2 framework

- JavaScript for Automation (JXA) payload for macOS

- Poseidon is a Golang payload for macOS

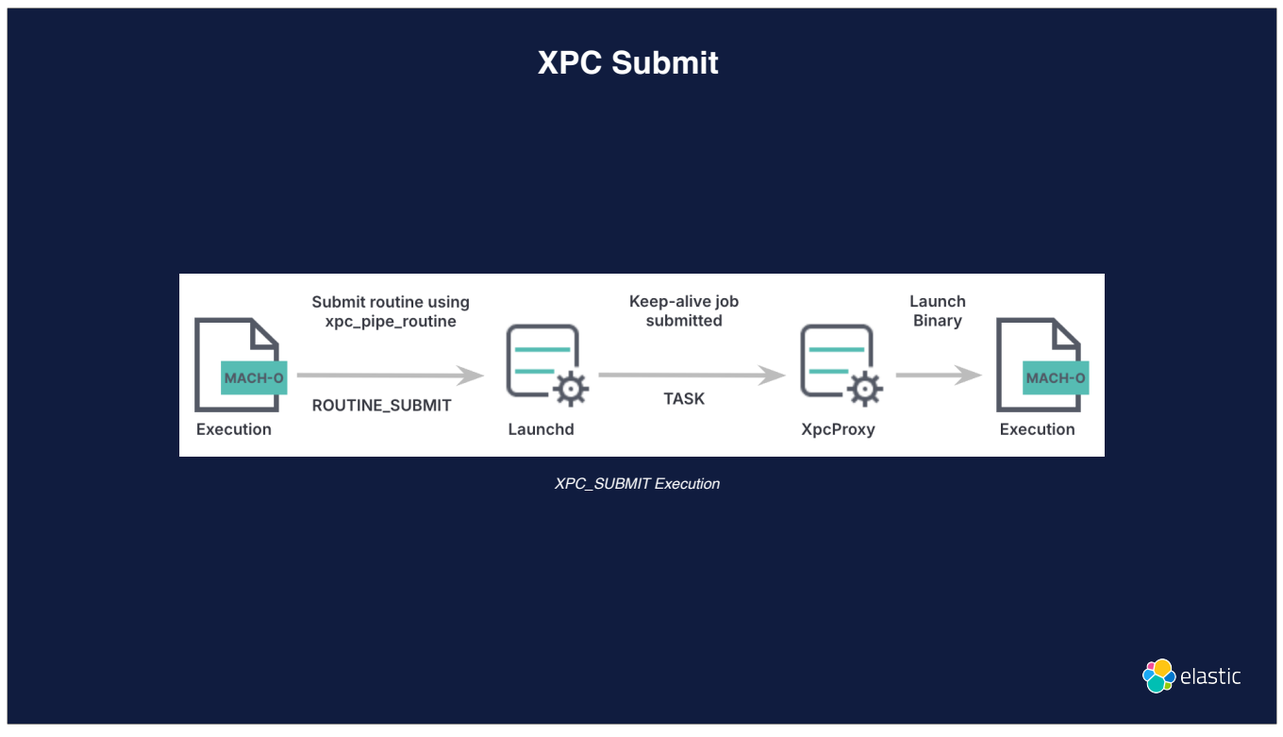
elastic

Quick lowdown on Mythic and these agents if you aren't familiar with them.

Mythic is cross-platform containerized C2 framework that is actively maintained and my personal favorite to work with created by Cody Thomas, who has spoken at this conference and is here, he's also known by his handle Its-A-Feature, amazing work Cody

Apfell is an agent built for macOS compatible with the Mythic C2 framework and is written in JXA originally created by Cody as well

Poseidon is an agent also compatible with Mythic C2 and is written mostly in Golang and supports both MacOS and Linux originally created by Chris Ross also known as Xorrior who has spoken at this conference before



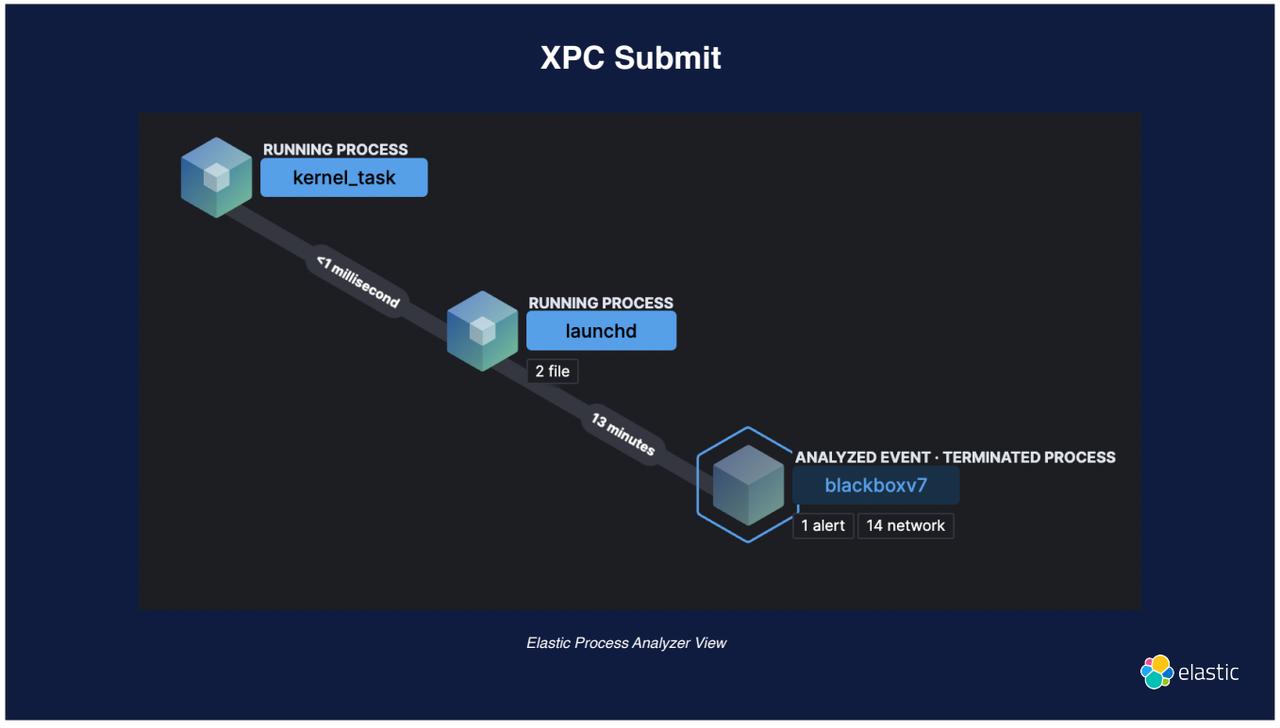
XPC_SUBMIT Execution

Let's start with the XPC submit command provided by the Poseidon agent.

The commands functionality allows the user to execute any binary on the system via XPC without requiring a backing Plist. It does this using the little known ‘xpc_pipe_routine’ function and ROUTINE_SUBMIT routine to submit a program for launchd to execute on its behalf

If you aren’t familiar with XPC its similar to RPC on Windows. Interprocess communication mechanism on macOS

This a defense evasion technique as it can break the execution chain and make the execution of the desired binary appear normal to users and defenders



I tested the command by executing a very basic, stripped down and obfuscated “dropper” Poseidon payload to then download my full Poseidon payload, with all the commands, via Curl then use this XPC submit command to execute my “second stage” payload and when you look at the execution of the second stage payload this is all you see.

Launchd executing the payload like you would see if a user double clicked it or the system executed it after login or reboot or something.

Seeing this I then set about analyzing the events collected from the techniques execution in order to assess if the behavior is observable and therefore detectable

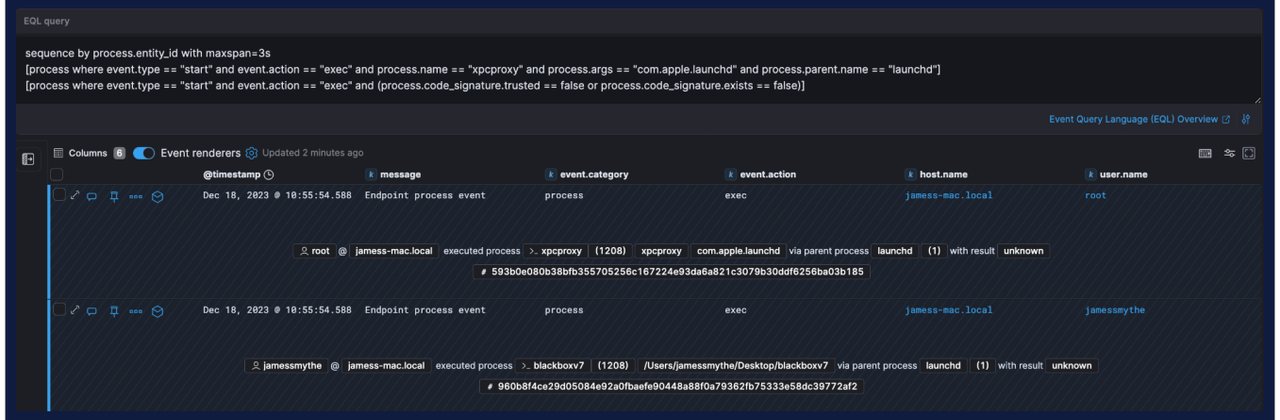
And when I looked closely at the process execution events I saw something very interesting. Remember when I said threat actor activity usually stands out against established norms?

I observed Launchd spawning its helper XpcProxy to invoke itself. I had never seen that before. Normally you will see XpcProxy invoke the program or service it is tasked with setting up and initiating but not in this case.

I then noticed that the process ID of the xpcproxy in this strange case matched the process ID of the full Poseidon payload I had tasked Launchd with executing

XPC Submit

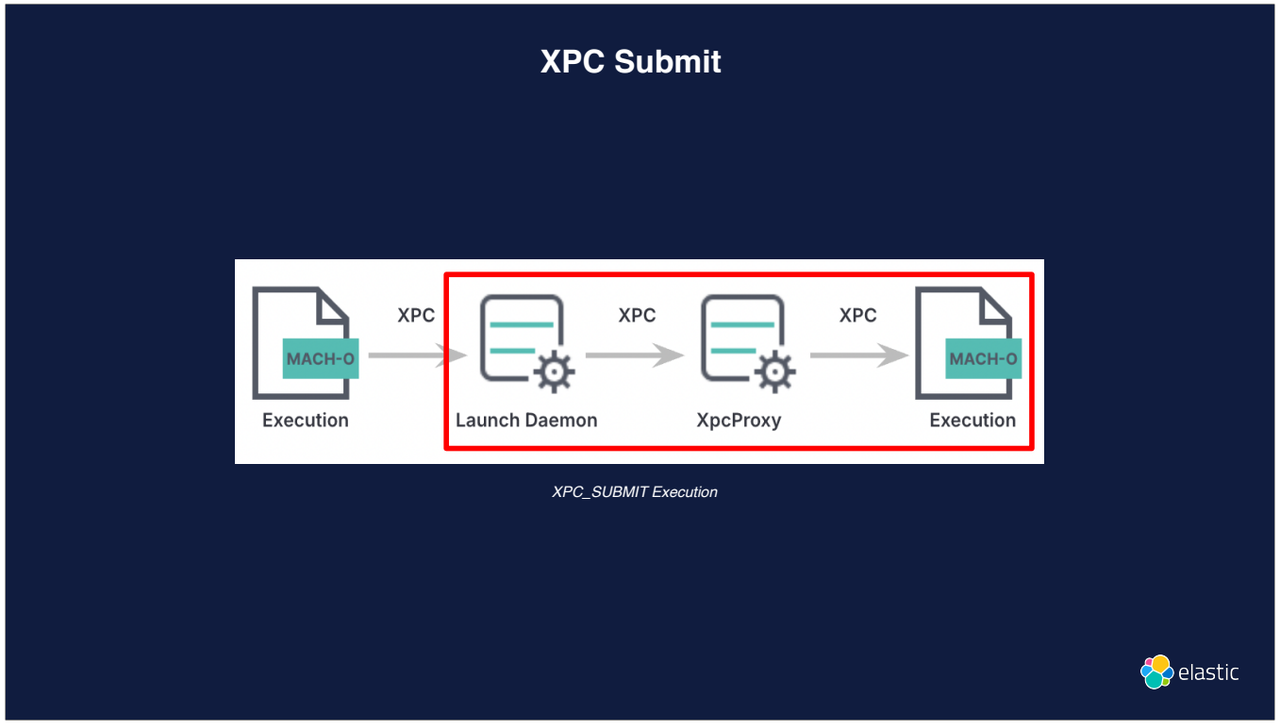
Unsigned or Untrusted Binary Launch via XPC_PIPE_ROUTINE



I realized that I could fingerprint this unique activity creating a Behavior Rule to detect it

We can sequence by the unique process tree created then looking for the unusual XpcProxy invocation of Launchd itself, which is an artifact of using the XPC_PIPE_ROUTINE command, followed immediately by the execution of an unsigned or untrusted binary, in this case my full Poseidon payload

Voila “Unsigned or Untrusted Binary Launch via XPC_PIPE_ROUTINE” tripwire set

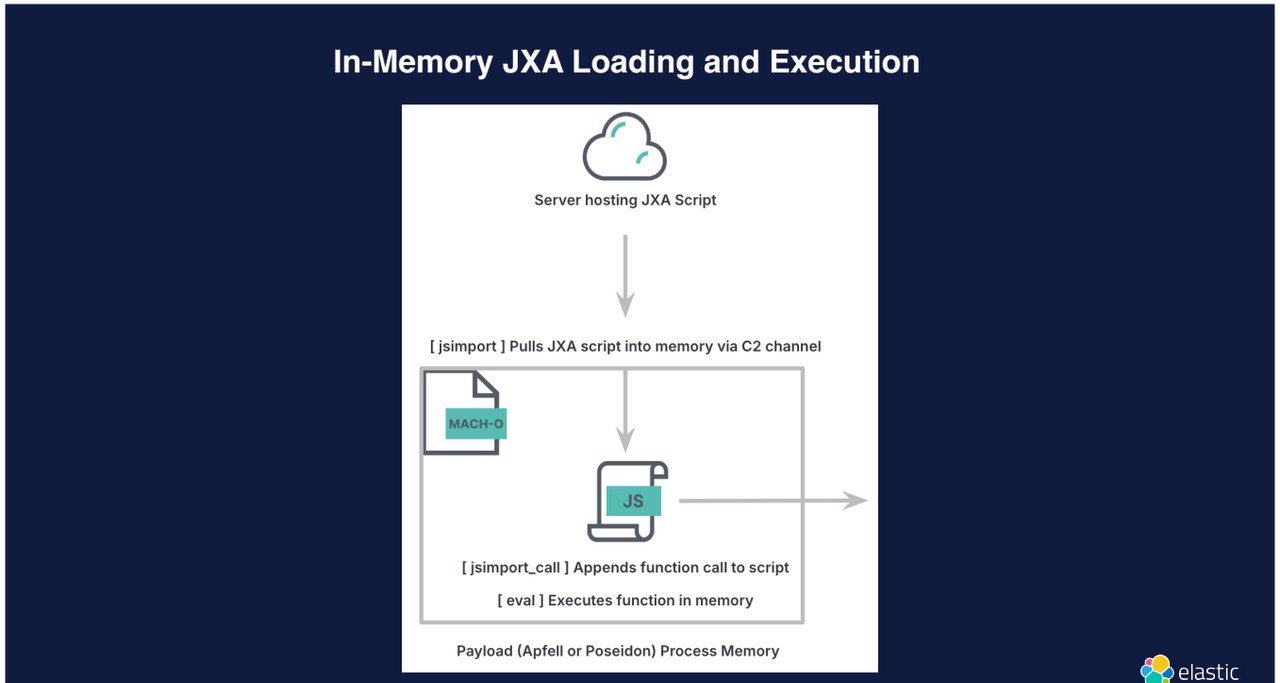


XPC_SUBMIT Execution

We are detecting this technique by focusing on the outcome of it.

We could of course create additional rules that look for this technique being utilized to execute Curl or Osascript etc. instead of just an unsigned or untrusted binary.

Let's move on to another very interesting technique though



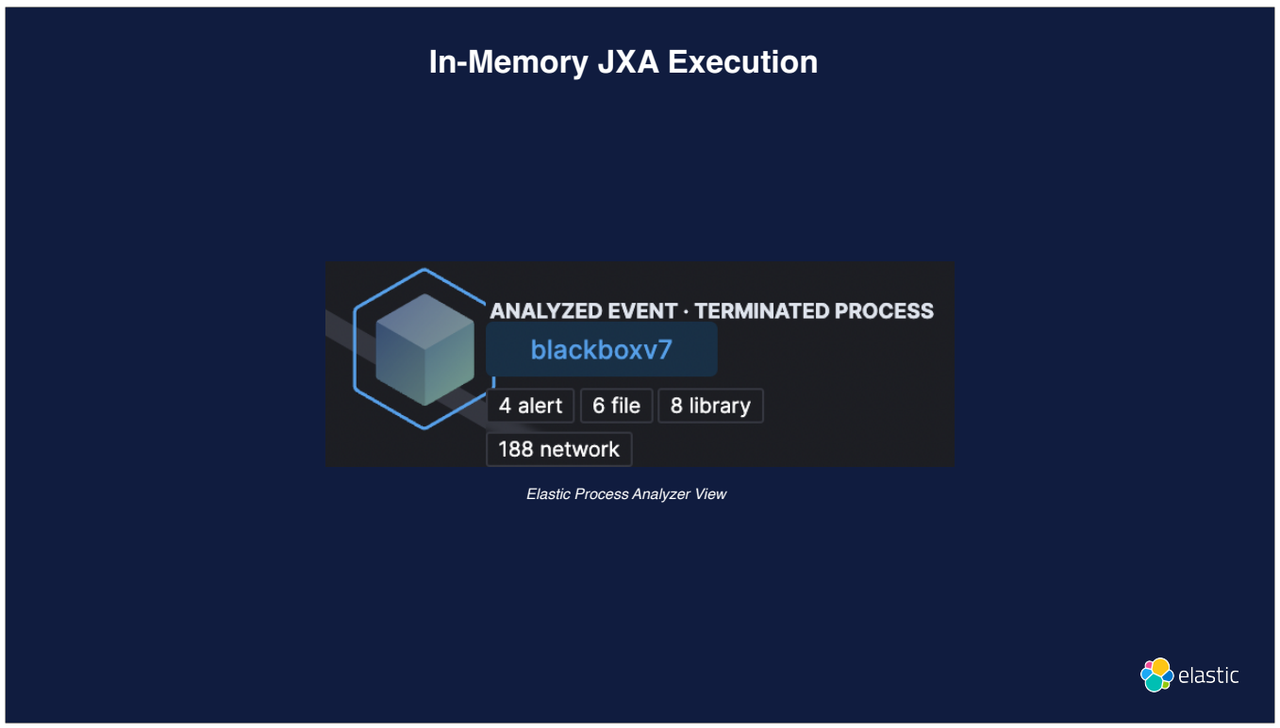
In-Memory JXA Load & Execute

Probably one of the most effective and cool evasion techniques out there today. In-Memory JXA loading and execution. This is implemented by both Apfell and Poseidon agents and is made up of two separate commands.

The JsImport command allows you to specify a JXA script or payload and it loads the script into memory assigning the contents of it to a variable

For those that don’t know JXA stands for JavaScript for Automation and is a scripting language that allows you to automate tasks and control applications using JavaScript serving as an alternative to AppleScript

The JsImport_call command allows you to specify a function to call from within the JXA script that was imported in order to execute the contents of the script via eval



Elastic Process Analyzer View

I loaded into my Poseidon payloads memory a situational awareness JXA script called Health Inspector also created by Cody Thomas using jsimport and then used jsimport_call to call the main function of this script executing its contents with the output of the script returned to my Mythic C2 console

Since all of this takes place within your payloads process memory you would think there would be absolutely no evidence no Behaviors to observe and in almost all other circumstances you would be right

But in a recent version of Elastic Defend we debuted a first of its kind custom Dylib Load event complete with Dylib Code Signature data, Process and Process code signature data with the ability to sequence across all other data sources and events

I decided to take a look at the libraries my payload was loading into memory just out of curiosity

In-Memory JXA Execution

dll.path	process.name
/System/Library/Components/AppleScript.component/Contents/MacOS/AppleScript	blackboxv7
/System/Library/Components/JavaScript.component/Contents/MacOS/JavaScript	blackboxv7
/System/Library/Frameworks/Foundation.framework/Versions/C/Resources/BridgeSupport/Foundation.dylib	blackboxv7
/System/Library/Frameworks/CoreFoundation.framework/Versions/A/Resources/BridgeSupport/CoreFoundation.dylib	blackboxv7
/System/Library/Frameworks/CFNetwork.framework/Versions/A/Resources/BridgeSupport/CFNetwork.dylib	blackboxv7
/System/Library/Frameworks/AppKit.framework/Versions/C/Resources/BridgeSupport/AppKit.dylib	blackboxv7
/System/Library/Frameworks/CoreGraphics.framework/Versions/A/Resources/BridgeSupport/CoreGraphics.dylib	blackboxv7
/System/Library/ScriptingAdditions/StandardAdditions.osax/Contents/MacOS/StandardAdditions	blackboxv7

Elastic ESQL

elastic

Two libraries stood out to me. The JavaScript library and the Scripting Additions Library. I immediately thought they might to be related to the JXA execution.

So I tested again with a fresh payload in a new VM and sure enough those libraries were not loaded until I executed the jsimport and jsimport_call commands.

I delved deeper into the code to attempt to understand what exactly might be causing those libraries to load

I wanted to do this in order to understand the technique better but also to validate my theory that the combination of those two libraries being loaded in quick succession could be a fingerprint for this activity.

In-Memory JXA Execution

jsimport

```
script = ObjC.unwrap($.NSString.alloc.initWithDataEncoding(script_data, $.NSUTF8StringEncoding));
```

/System/Library/Components/JavaScript.component/Contents/MacOS/JavaScript

elastic

Looking at the code that implements the loading of the JXA into memory I found that the Objc.unwrap method called is part of the JXA environment and used to convert the JXA script content to a string assigned to the script variable.

Going deeper I then discovered that the JavaScript library is what includes and implemented the Objective See bridge containing this method

In-Memory JXA Execution

jsimport_call

/System/Library/Components/JavaScript.component/Contents/MacOS/JavaScript

```
let output = ObjC.deepUnwrap(eval(jsimport + "\n " + command_params['command']));
```

/System/Library/ScriptingAdditions/StandardAdditions.osax/Contents/MacOS/StandardAdditions

elastic

Looking at the code that implements the jsimport_call to actually execute our JXA script we now know that the Objective See method call loads the JavaScript library into memory.

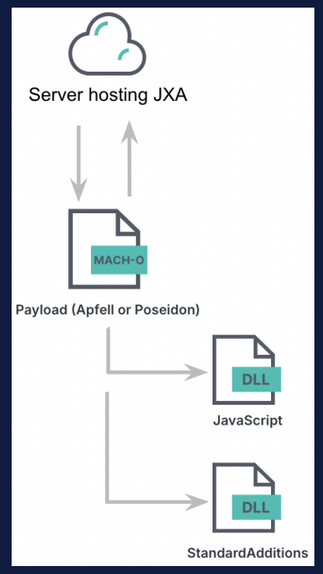
I then noted that the eval function was being used to actually evaluate the JavaScript code represented as a string. It parses the string and executes it as code.

Doing some digging I discovered that it was the use of eval that causes the Standard Additions library to be loaded

When you use eval, the JXA runtime needs to ensure that the execution environment is fully prepared to handle any code that might be evaluated.

The Standard Additions library includes a set of essential commands and handlers commonly used in scripts, such as file operations, user interactions, and system controls.

In-Memory JXA Execution



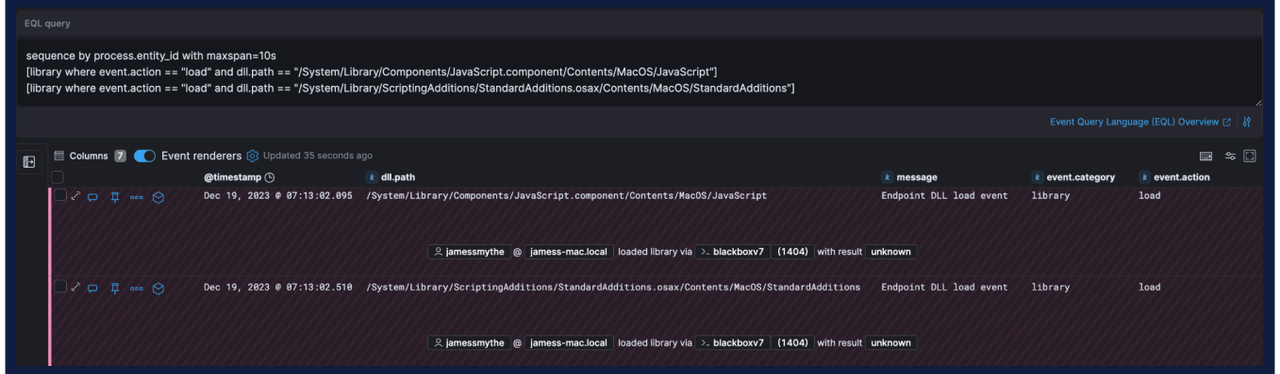
In-Memory JXA Load & Execute

elastic

Summarizing what we've learned this is the behavior this technique exhibits at a high level

In-Memory JXA Execution

In-Memory JXA Execution via ScriptingAdditions



elastic

Looking at our Dylib load events we can see the Behavior we've identified with the JavaScript library being loaded followed immediately by the Standard Additions library by our Poseidon payload

We can now create our Behavior rule that fingerprints this specific activity.

Sequencing by a unique process tree within a 10s timespan where the JavaScript library is loaded followed by the Standard Additions library

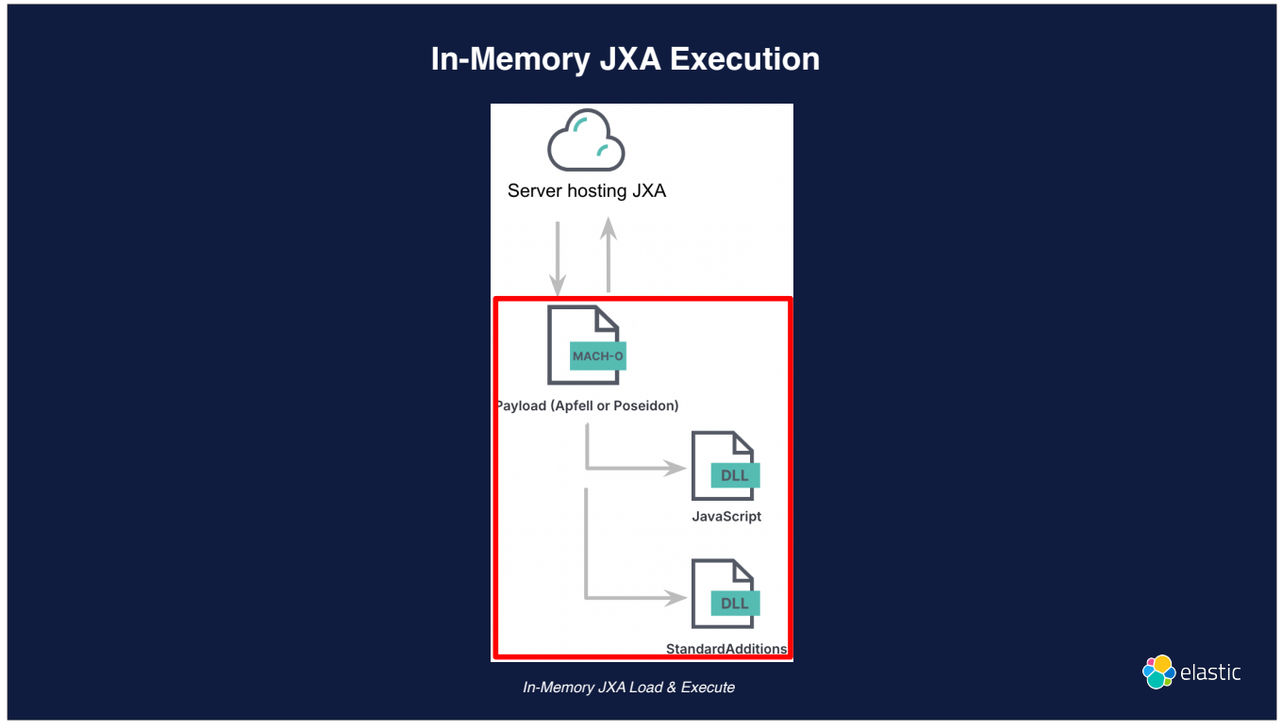
Now I had the same thought you might be having right now which is This has got to be common right? Noisy?

So I ran the query across the vast expanse of our macOS telemetry data and got nothing

I then deployed the Behavior rule in diagnostic across our Endpoints and encountered extremely minimal false positives

Tested the technique again and the rule fired

Now something to note is that it will only fire the first time this technique occurs on a system because then the libraries are loaded into memory and it doesn't need to keep loading them everytime you import and execute a script but that's all we need to detect the activity and kill the process



In-Memory JXA Load & Execute

Our Behavior rule detects this activity and if detected the responsible process and its tree will be killed.

I've tested a variety of implementations out there that utilize this technique in different languages and formats and this rule detects every single one of them

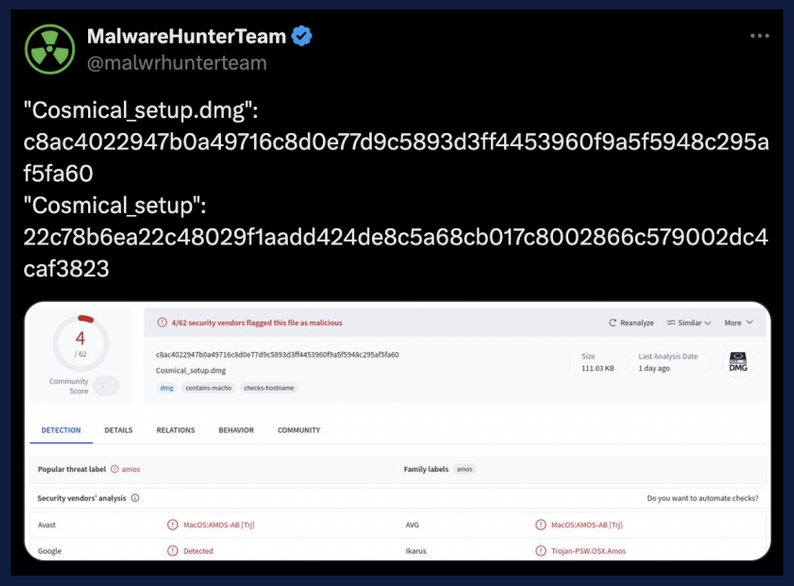
It is an unavoidable result of the code required to implement it and apparently not common to be implemented by normal developers

We’ve done some detection engineering work now lets look at how effective these Behavior rules are in the wild and how they have allowed me and my team at Elastic to stay ahead of the bad guys

We will take a look at two case studies here that I feel like provide very good examples of why these rules can be so powerful and frustrating for the adversary

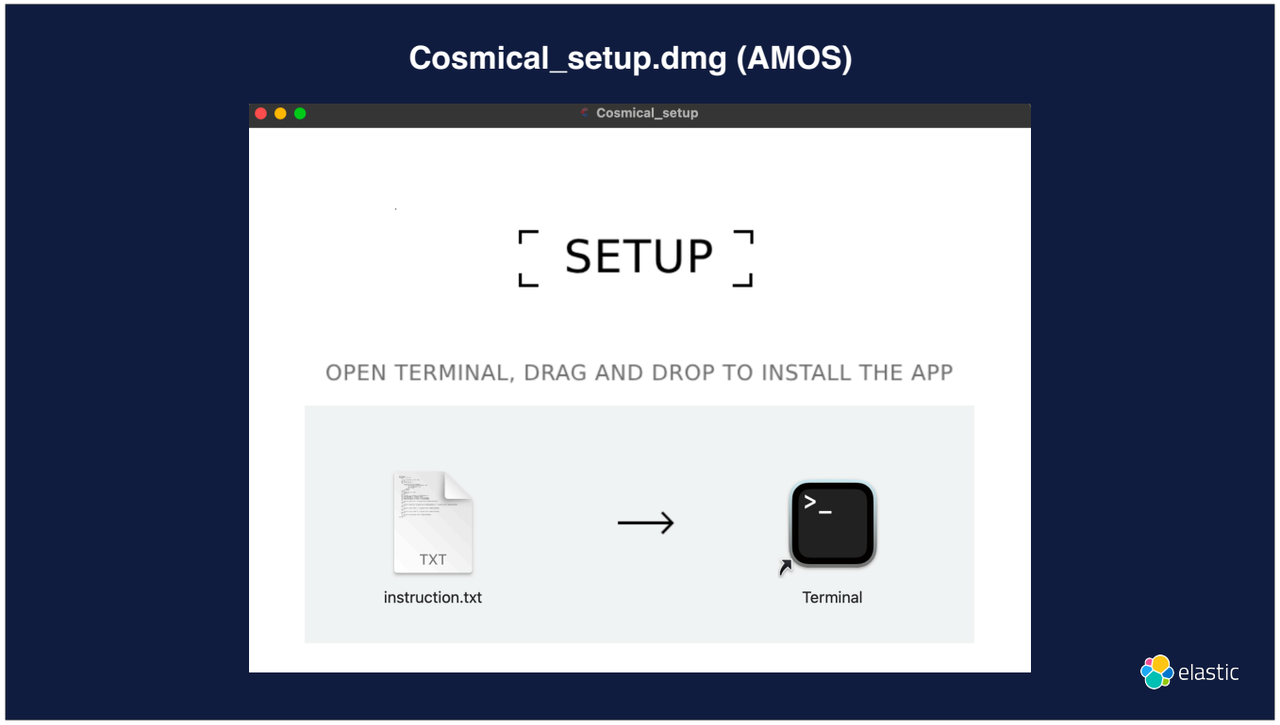
For case study 1 we are going to talk about AMOS. AMOS is an information stealer developed to target macOS and is probably the most prolific macOS information stealer in the wild today.

Cosmical_setup.dmg (AMOS)



We are going to focus on this recent, new AMOS variant surfaced by MalwareHunterTeam on X.

Only 4 generic detection on VirusTotal with none of those being macOS EDR vendors.



I detonated the sample and it prompts the user to drag a “.txt” file, which is actually a bash script which executes Osascript to execute AppleScript, anyway drop that into Terminal and execute it then enter your password into the Osascript password prompt to give it access to your data

Whats interesting is that this is a new method of user execution the threat actor created in order to bypass the right click to open Gatekeeper bypass user execution Apple decided to lock down in a the most recent version of macOS

I went to my Elastic Alerts dashboard to see if anything was detected

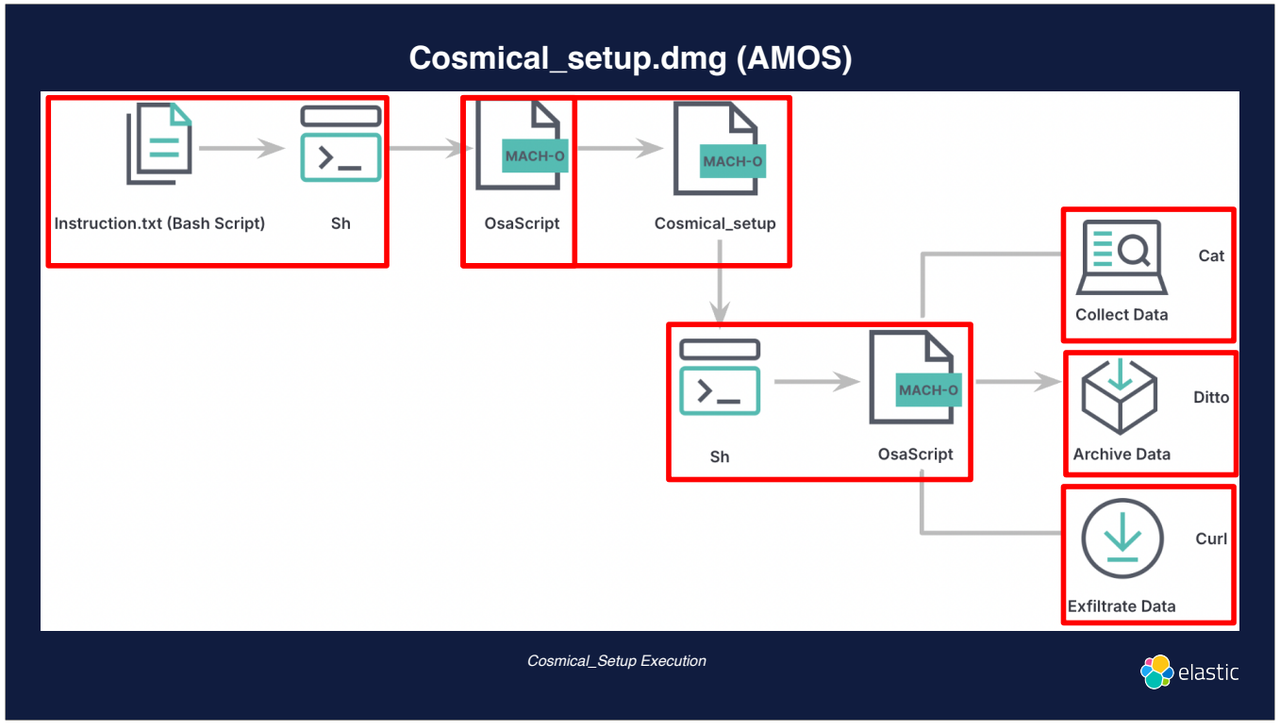
Rule
Malicious Behavior Detection Alert: Execution of Non-Executable File via Shell
Malicious Behavior Detection Alert: Suspicious Script or Process Execution from Mounted Device
Malicious Behavior Detection Alert: Potential Credentials Phishing via Osascript
Malicious Behavior Detection Alert: Dscl Execution via Osascript
Malicious Behavior Detection Alert: Dscl Execution via Osascript
Malicious Behavior Detection Alert: User Keychain DB Access by Osascript
Malicious Behavior Detection Alert: Web Browser Credential Data Accessed by Osascript
Malicious Behavior Detection Alert: Web Browser Credential Data Accessed by Osascript
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Crypto Wallet File Access via CommandLine
Malicious Behavior Detection Alert: Potential Data Exfiltration via Curl

Elastic Alerts Dashboard

Turns out almost the entire execution chain including the new method of user execution dragging the bash script disguised as a text file into Terminal.

All of these Endpoint Behavior detections were created either from past InfoStealer samples or completely different malware samples altogether providing cohesive coverage regardless of the fact that our YARA signatures like everyone else's did not detect the sample

And if my policy had been in prevent mode the malware would have been quarantined and killed the moment the user attempted to execute the ".txt" file with Terminal



We can detect them at every stage of the execution chain and even if they adapt to avoid one they will get caught by another.

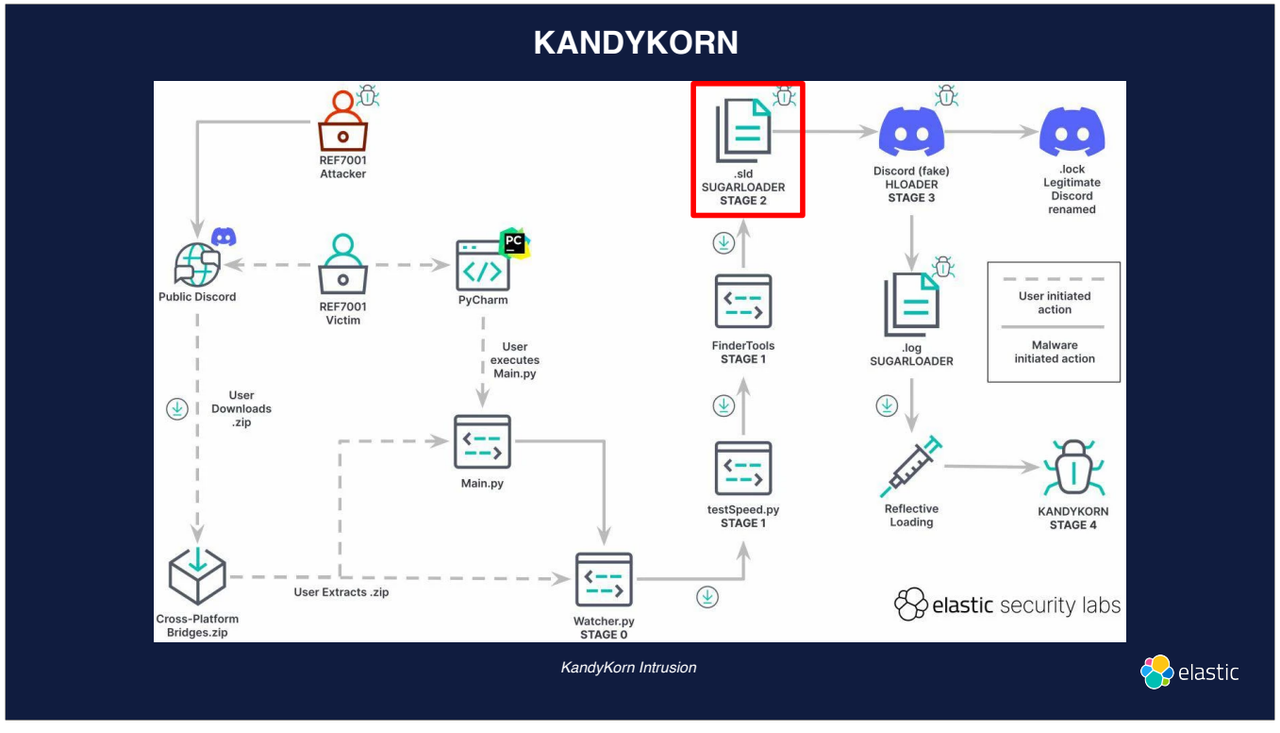
They have to dedicate immense time and effort in order to evade.

And every stealer I test against so far the entire execution chain of events is detected with me having to add maybe one or two more rules here and there

No matter what they do to evolve Behavior rules are there and they are unavoidable. Payload agnostic.

Next, on the opposite end of the spectrum, we are going to take a look at one of the most full scope and sophisticated intrusions against macOS to date KandyKorn discovered last year by me and written up with the help of my awesome colleagues at Elastic.

Little known fact I actually discovered the intrusion on the very last night of this conference last year in Spain.



KandyKorn Intrusion

elastic

These threat actors, who we believe to belong to the DPRK specifically a group dubbed Bluenoroff, were targeting Blockchain Engineers specifically on Discord public blockchain channels.

They provided a Blockchain arbitrage bot to try out in the form of a Python application.

Once executed the application proceeded to download two additional Python dropper scripts which ultimately ended in a hidden loader binary being dropped which was responsible in delivering and executing the final payload, we deemed KandyKorn, in memory via Reflective loading.

This intrusion was premeditated, tested, they knew what they were doing and went out of their way to avoid being detected. The payloads were custom developed. None of the Python components or payloads were on VirusTotal or anywhere else.

And without Behavior rules specifically one single Behavior rule I never would have discovered this intrusion and we at Elastic would not have been able to stop it and this campaign and campaigns like it would have continued undetected

Thanks to a single Behavior rule I created one year prior to the intrusion I was able to detect them at this stage and engage with the company in order to investigate, remediate and analyze this intrusion

KANDYKORN

rule.name	process.executable	file.name
Reflective Binary Load	/Users/Shared/.sld	NSCreateObjectFileImageFromMemory-P3JF36gx

EQL query

```
file where event.type != "deletion" and
file.name : "NSCreateObjectFileImageFromMemory-*" and
not process.name == null
```

https://slyd0g.medium.com/understanding-and-defending-against-reflective-code-loading-on-macos-e2e83211e48f

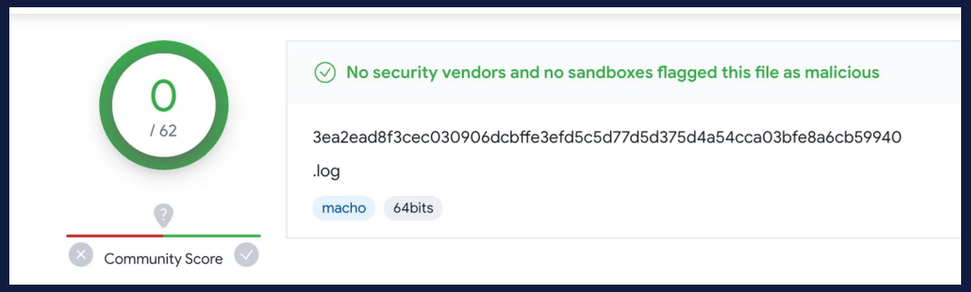
elastic

I detected them ironically attempting to do the single most advanced defense evasion technique they utilized “Reflective Binary Loading”.

Reflective loading involves allocating then executing payloads directly within the memory of the process, vs creating a process backed by a file path on disk

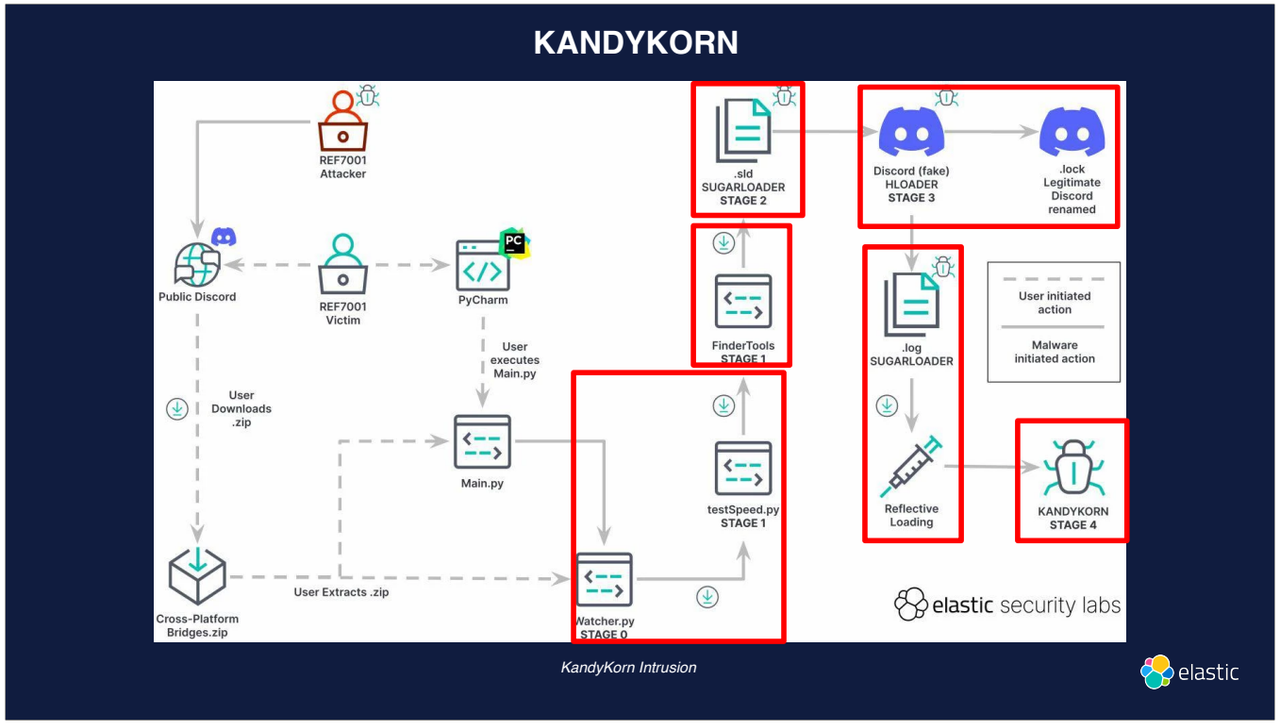
I read researcher and Red Teamer Justin Bui’s write-up on macOS reflective code loading in 2022, tested his tooling and created a simple Endpoint Behavior Rule looking for the resulting artifact the specific implementation of this technique exhibits on macOS

KANDYKORN



elastic

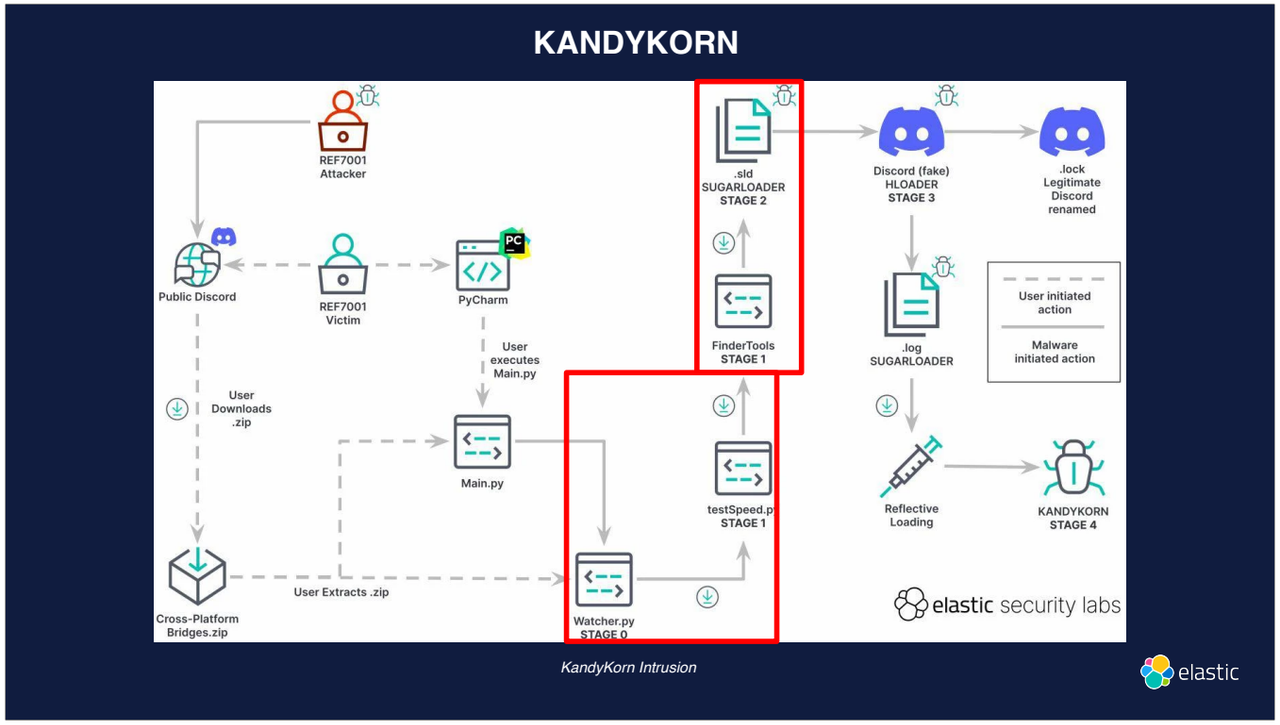
When we did upload the payloads to VirusTotal no one detected them in any way shape or form. As you might expect.



KandyKorn Intrusion

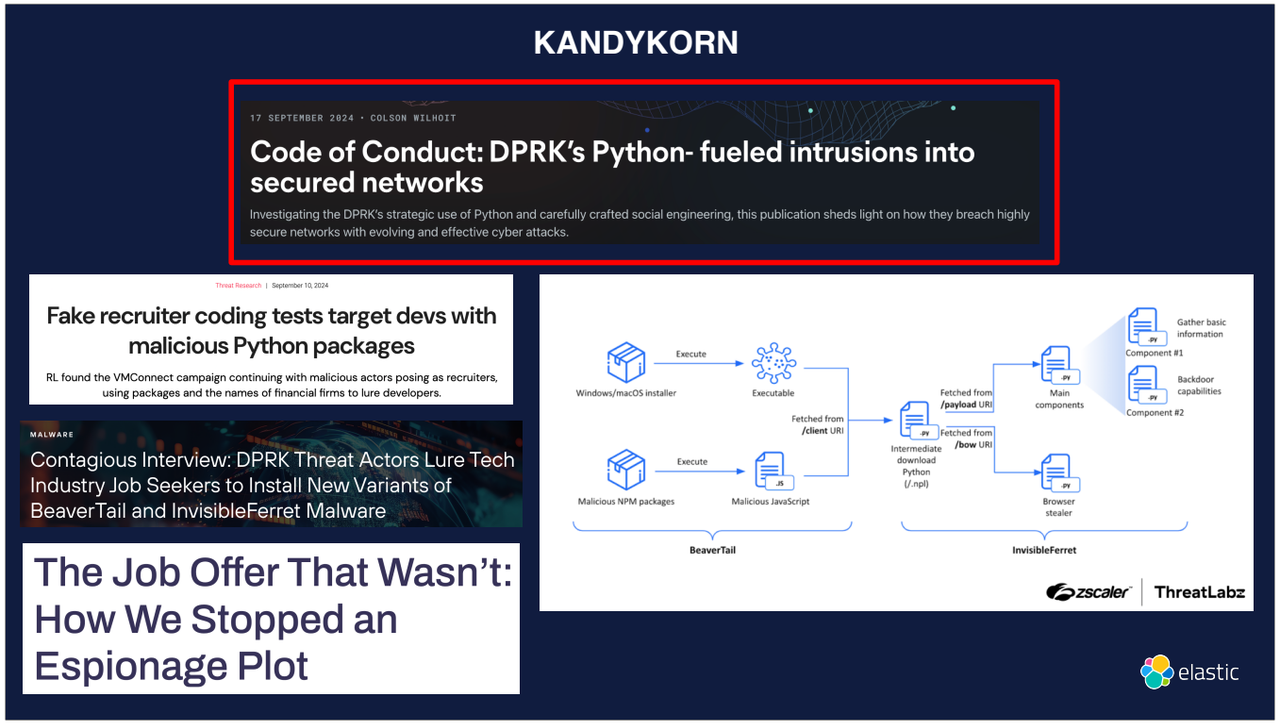
Of course now we detect the full scope of this intrusion in all aspects to include YARA, Mach-O model and Behavior rules.

I've added a whole slew of Endpoint Behavior rules as a result of this intrusion but it only took one to sink it



KandyKorn Intrusion

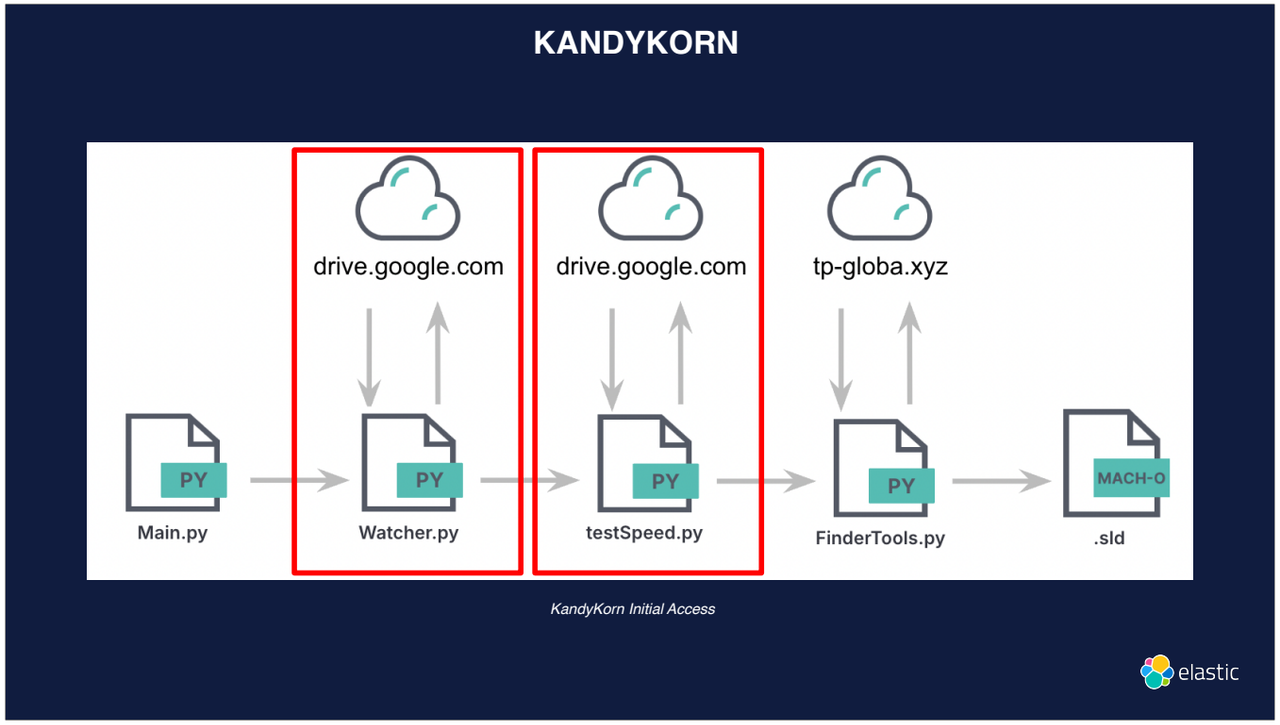
The ability to detect the Python initial access and droppers using Endpoint Behavior Rules has come about recently though.



The DPRK has been utilizing Python applications and scripts as their tooling of choice when it comes to gaining initial access to victim networks.

There are specific reasons for this and I've recently blogged about the evolution of these lures. You can check that out on our Elastic Security Labs page.

You can see additional write-up headlines listed here as well that highlight newer campaigns using Python based initial access lures akin to the one used in the KandyKorn intrusion



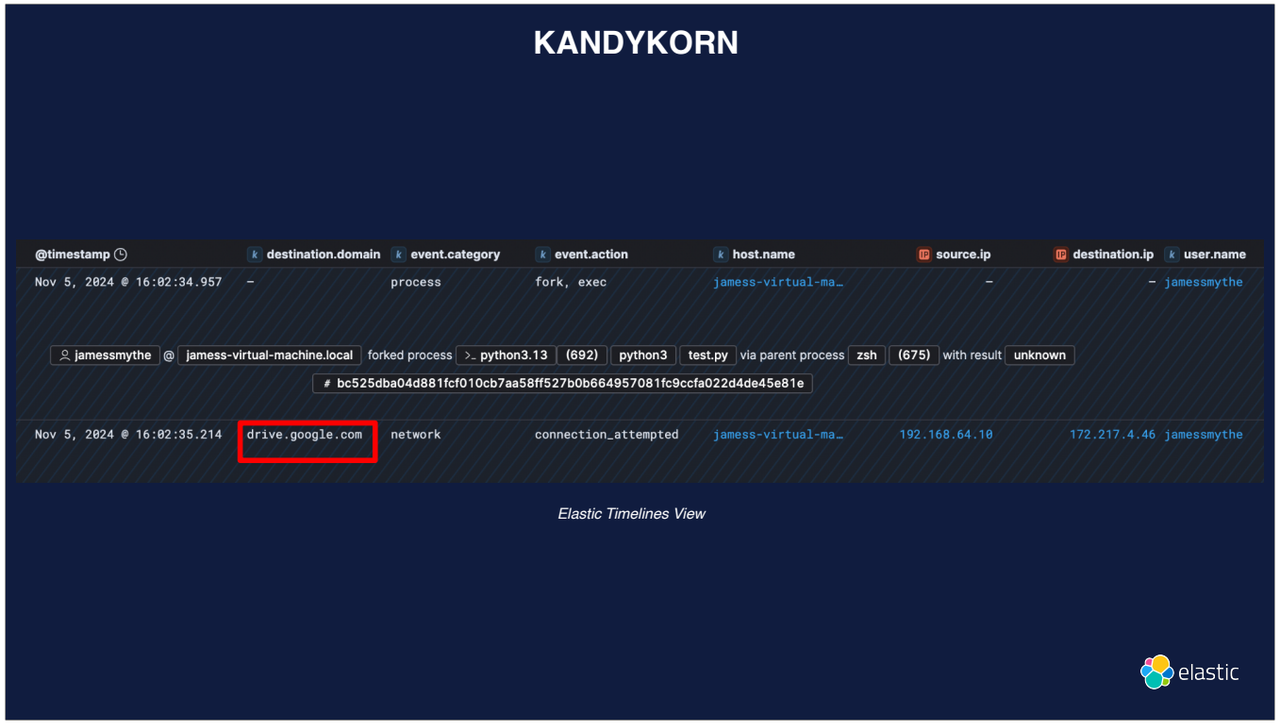
KandyKorn Initial Access

In the case of KandyKorn the initial Python application reaches out to Google Drive to obtain a Python script called testSpeed.py which itself is a dropper that reaches out to Google Drive to obtain yet another Python script dropper named FinderTools.py which is what finally delivers a binary payload

Side note file hosting and other cloud hosting services like Google Drive are effective means of delivering dropper payloads or scripts as the sites are often globally whitelisted and files hosted can be quickly removed all via API calls

In our recent release of Elastic version 8.16 we have enhanced our network events with the destination domain field and are currently testing dedicated DNS events

With this new field we have the ability now to detect and prevent many of these script based initial access attacks

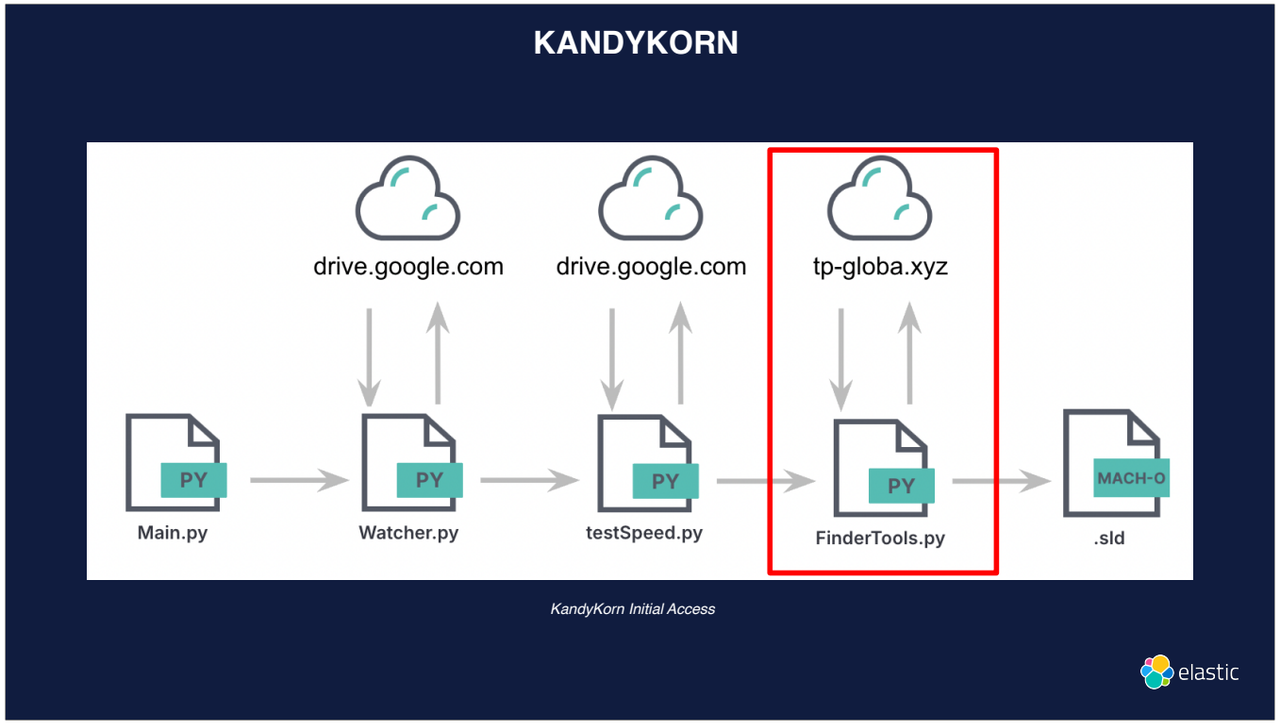


Elastic Timelines View

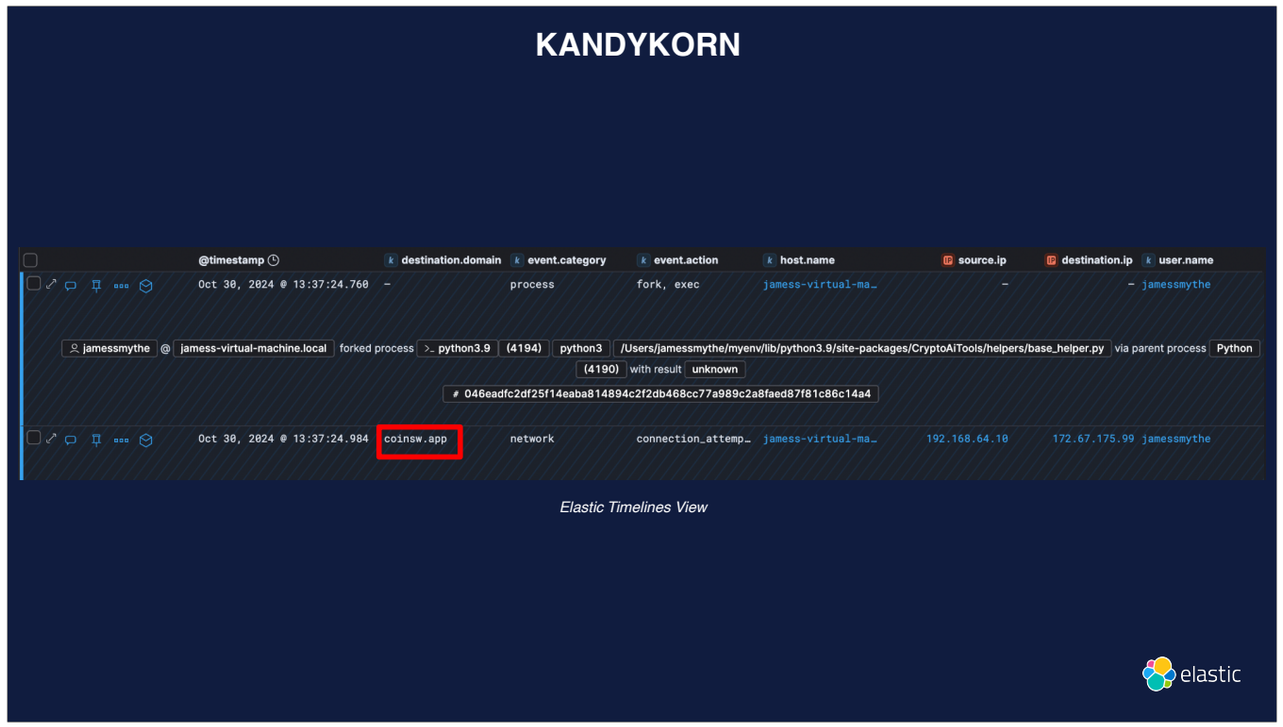
I’ve created a new Endpoint Behavior rule and others like it that look within unique process tree’s within a short window starting with the execution of a scripting languages like Python followed by an outbound network connection to a Google Drive and other Cloud hosting platforms like it

You can see in the screenshot here this new rule “Script based Initial Access via Cloud Hosting” detecting a Python script replicating the one used in KandyKorn to reach out and attempt to download a script or binary from Google Drive.

FYI hosting binaries or Python scripts on Google Drive is in no way normal and I’ve never see a developer or admin do this or need to do this in the wild



Like we've mentioned the last Python dropper did not reach out to Google Drive but to a custom C2 .xyz domain in order to grab the loader payload



Elastic Timelines View

I’ve created another new Endpoint Behavior rule “Network Connection to Suspicious Top Level Domain” that looks within unique process tree’s within a short window for execution followed by an outbound network connection to a suspicious top level domain.

Domains seen used in previous malware campaigns or abnormal domains.

In the screenshot here you can see this rule detecting one of these Python application lures in its attempt to download a second stage payload from a .app domain

The Future is Now. Where can we take Behavior Detections from here?

There are a couple of really cool things we are doing right now that really elevate the power of Behavior Detections and provide additional angles at which to approach threat detection

Sequencing Behavior Rules

EQL query

```
sequence by agent.id
[any where event.dataset == "endpoint.alerts" and rule.name == "Execution of Self-Signed Binary from Volume Mount"]
[any where event.dataset == "endpoint.alerts" and rule.name like ("Potential Credentials Phishing via Osascript", "Terminal Window Hidden or Closed via Osascript")]
```

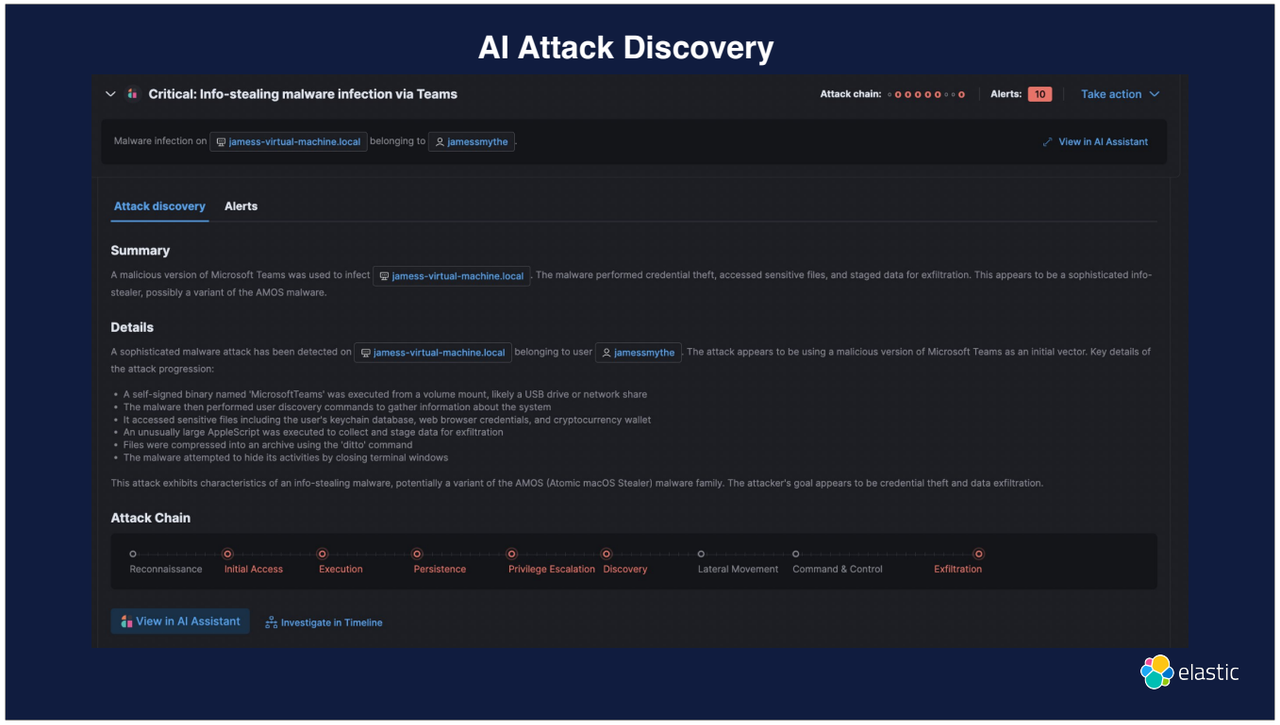
Potential Atomic Stealer

elastic

Once you implement enough Behavior Rules you can start sequencing the rules themselves to create malware Behavior Signatures

I’ve done this in my Elastic testing stack to quickly identify malware families by Behavior and in our telemetry stack to quickly surface malware and threat actor specific attacks

Pretty cool.



A new feature in Kibana now is AI Attack Discovery.

You can connect Kibana into any of the hosted LLMs like Bedrock, Gemini, or OpenAI, use your own hosted LLM or bring your own local LLM

AI attack discovery works by analyzing Behavior Rules and their associated data to identify threats and connect the dots taking the work of several analysts or what would take one analyst several hours and boiling that down to minutes. Each "discovery" represents a potential attack and describes relationships among multiple alerts to tell you which users and hosts are involved, how alerts correspond to the MITRE ATT&CK matrix, and even which threat actor it might be attributed to.

It does this in minutes and sometimes seconds. Providing you this cohesive breakdown attack summary, details and attack chain to include all associated alerts which you can import directly into a case and action

As a researcher this tool is super useful as well allowing me to gain a comprehensive awareness of what a piece of malware is doing very quickly without all the manual analysis

User and Entity Behavior Analytics (UEBA) Rules

```
name = "First Time Seen Removable Device"
references = [
    "https://winreg-kb.readthedocs.io/en/latest/sources/system-keys/USB-storage.html",
    "https://learn.microsoft.com/en-us/windows-hardware/drivers/usbcon/usb-device-specific-registry-settings",
]
```

```
type = "new_terms"

query = '''
event.category:"registry" and host.os.type:"windows" and registry.value:"FriendlyName" and registry.path:*USBSTOR*
'''
```

elastic

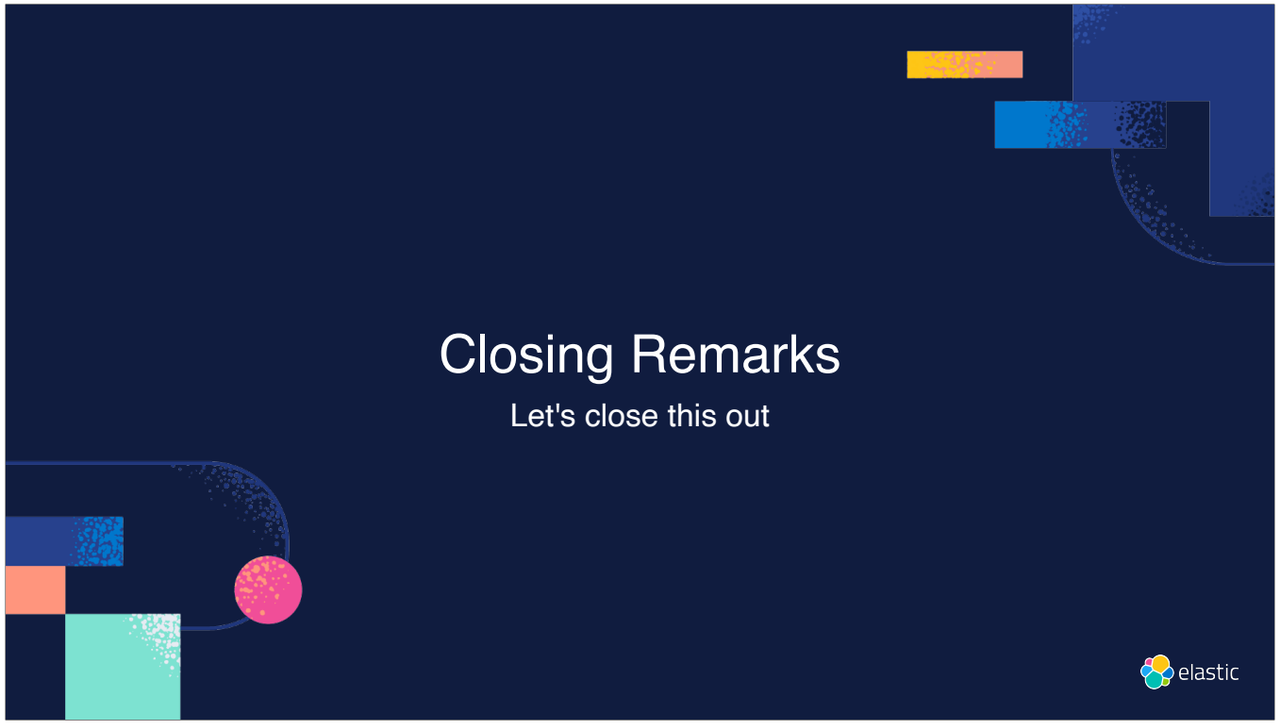
We can also create UEBA type rules using language features found in Kibana like new_terms that allow us to trigger a Behavior Rule on the first occurrence of the Behavior being seen on a system or by a user

Endpoint Behavior Reputation Rules

```
name = "Malicious Reputation Binary Dropped via Archive Utility"
```

elastic

We can also trigger Behavior rules and check the binary or file hash’s reputation against VirusTotal and other such platforms to identify and ensure we detect known or observed malicious processes, libraries or files that have already



Alright let's close this out

Elastic's Commitment

I want to reaffirm Elastic's and my commitment to our community, users and all of you. We will continue to push the boundaries of defense on macOS, impose cost on the adversary, burn campaigns and share what we find.

We are the only EDR/XDR solution that is open source and allows you download, use and test our product for yourself so don't take my word for it. Our open source nature is one of many reason I love working here.

I am obviously biased but I don't think there is a more powerful agent, platform, tool for Defenders and Researchers alike. Not only do we provide so much unique value but the value you can bring and things we enable you to do yourself extremely powerful.

Endpoint Protection Artifacts

README License Security

ATT&CK Navigator

Protections Artifacts

Elastic Security prevents ransomware and malware, detects advanced threats, and arms responders with vital context. It's free and open, ready for every endpoint.

Protections-Artifacts is the home of our detection logic (rules, yara, etc) for Elastic Security for endpoint. At Elastic, we believe that being open and transparent is critical for the success of us and our users. Check out our blog post if you are interested in additional background.

Directory

Below you will find the artifacts we have opened in this repository:

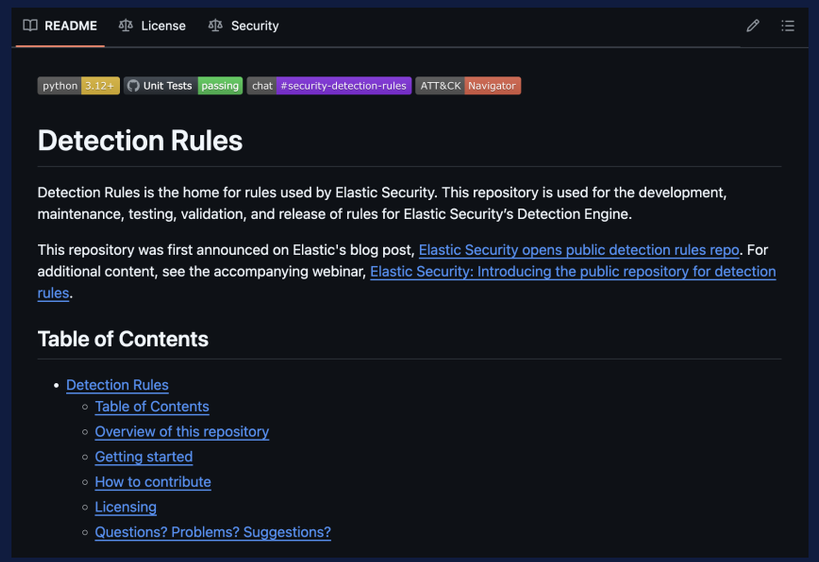
Folder	Description
behavior/	EQL based malicious behavior rules
yara/	Yara rules for malware protection
ransomware/	Elastic ransomware protection artifact

https://github.com/elastic/protections-artifacts

elastic

Speaking of Open Source you can find our Endpoint Elastic Defend production artifacts to include our production Endpoint Behavior rules, YARA ruleset and ransomware artifacts here in our Protections Artifacts Github repository

SIEM Detection Rules



https://github.com/elastic/detection-rules

elastic

Our SIEM detection rules are also available for you review here. You can contribute to these by creating a PR to critique a current rule, tune a current rule, or add your own rules.

Elastic Container Project

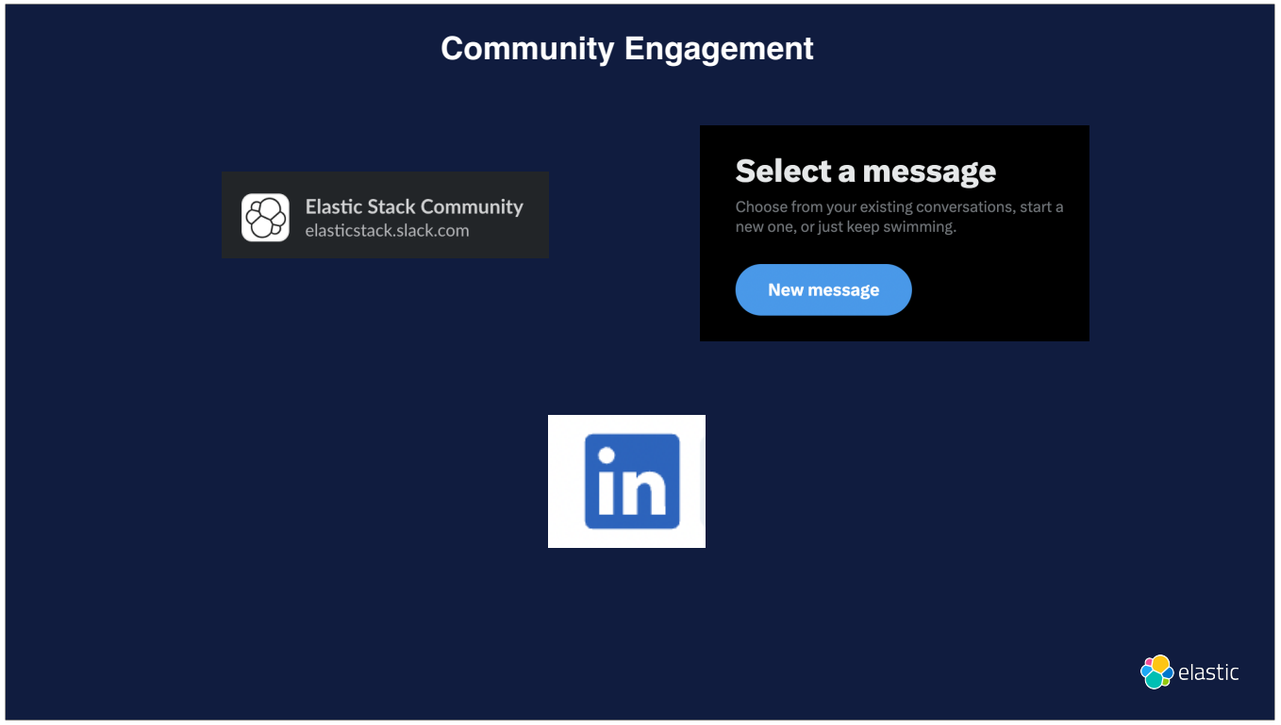
https://github.com/peasead/elastic-container

elastic

If you want to explore what we offer, all the cool features and things I've talked about today or want to use the stack like I do as a researcher and detection engineer.

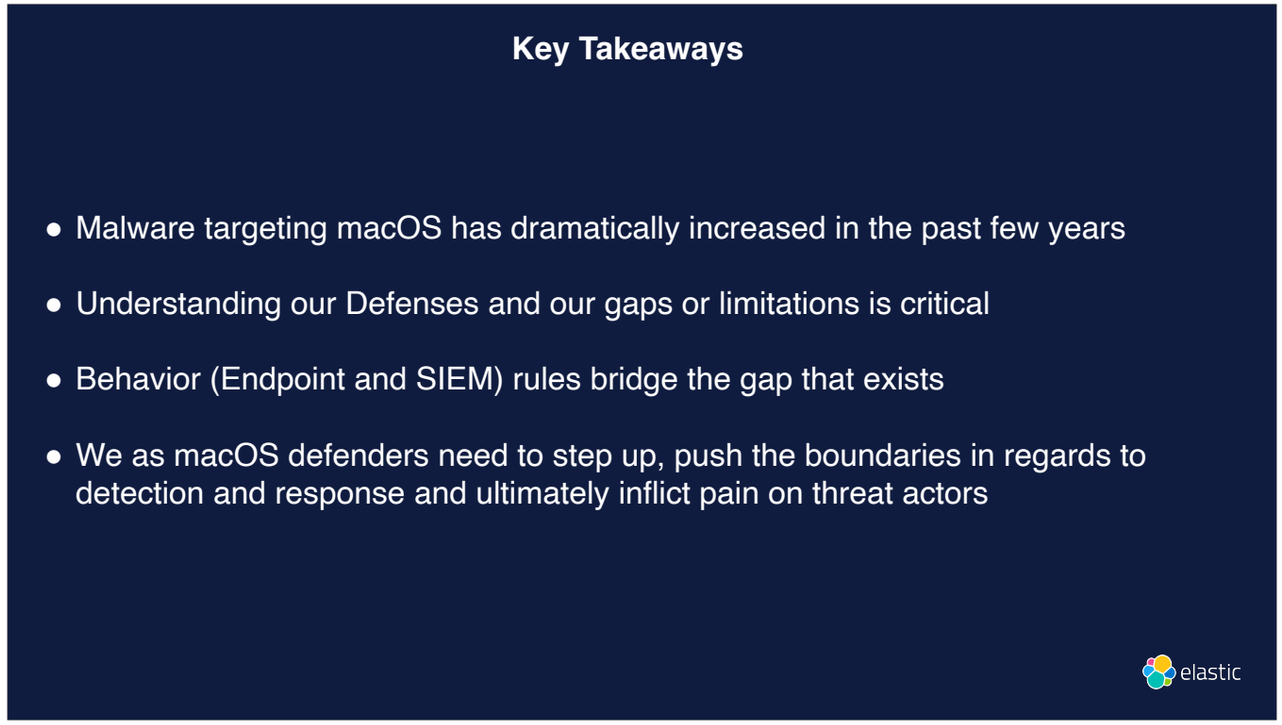
My colleague and I have created "The Elastic Container Project" which allows you to stand up a pre-configured 100% containerized, TLS secured Elastic Stack within minutes. You can enable the trial license as well to get all of our premium features.

Install an Elastic Agent in a VM and your off to the races.



If you want to reach me you can ping me on LinkedIN or X.

And of course we have our Elastic Stack Slack Community channel as well which is a great place to glean information or ask questions



Key Takeaways from this presentation:

- Malware targeting macOS has dramatically increased in direct relation to the increase in adoption
- Understanding our defense landscape along with our gaps and limitation is critical
- Behavior rules (Endpoint and SIEM) bridge the gap in our Defense Landscape and provide a powerful proactive defense mechanism unique to macOS
- We as macOS defenders need to step up, adapt and shape the future of defense

Thanks!