

FINDING BEACONS IN THE DARK

A Guide to
Cyber Threat Intelligence



BlackBerry Research and Intelligence Team

INTRODUCTION



Cobalt Strike provides adversary simulation and threat emulation software that is widely used by red teams and heavily abused by malicious threat actors. It has become a highly prevalent threat, employed by a vast number of Advanced Persistent Threat (APT) and cybercrime groups across the globe.

It is easy to see why this is the case, as it is fully featured and well-documented. From reconnaissance and spear-phishing to post-exploitation and covert communications, Cobalt Strike is feature-rich, well supported and actively maintained by its developers. Beacon, Cobalt Strike's primary payload, provides a wealth of features for attackers, which facilitate:

- Reverse shells and remote command execution
- Keylogging and screenshots
- Data exfiltration
- SOCKS proxying
- Pivoting
- Privilege elevation
- Credential and hash harvesting
- Port scanning and network enumeration

For a lot of legitimate as well as criminal organizations, leveraging Cobalt Strike can be cheaper and faster than developing their own tooling. At the time of writing, licensing starts at \$3,500 per license per year. If you are an unscrupulous bad actor who is using a cracked or leaked copy, the cost goes down to literally nothing!

From a threat intelligence or law enforcement perspective, Cobalt Strike's widespread use can often make the task of attribution more challenging, and the current upward trend in utilization is not showing any sign of decline.

Proofpoint researchers recently reported 161% year-over-year growth in the use of Cobalt Strike by cyber-criminals. It has become a perennial problem for security practitioners, requiring robust solutions that can both aid in providing defensive capabilities and enhanced threat intelligence.

Threat Post

Cobalt Strike Usage Explodes Among Cybercrooks

<https://threatpost.com/cobalt-strike-cybercrooks/167368/>

Proofpoint

Cobalt Strike: Favorite Tool from APT to Crimeware

<https://www.proofpoint.com/us/blog/threat-insight/cobalt-strike-favorite-tool-apt-crimeware>

On the defensive side of things, the best thing we can do to tackle the challenge of combating the rogue use of Cobalt Strike is to have solid processes in place. These processes need to be not only well thought out, but also driven by data. We have defined a robust Cyber Threat Intelligence (CTI) lifecycle that considers stakeholders for all products and services across the extended detection and response (XDR) solution space. Over the course of this book we'll guide you through our lifecycle and use Cobalt Strike as a practical hands-on case study.

You may ask, what is XDR? XDR is a fairly new term and one that a lot of folks are not yet familiar with. This is how IT consulting firm Gartner has defined it:

“XDR is a SaaS-based, vendor-specific, security threat detection and incident response tool that natively integrates multiple security products into a cohesive security operations system that unifies all licensed components.” - Gartner

At its core, XDR is a data ingestion and enrichment strategy. This means that it ingests telemetry from cyber-security products and services, as well as insights from threat intelligence teams and information from 3rd party sources. This data is then stored in a data lake, which is essentially a storage solution for raw data on any scale. The ingested data is then further processed to create additional context, which then drives intelligence-based threat-detection and correlation of incidents and alerts for all XDR products and services.

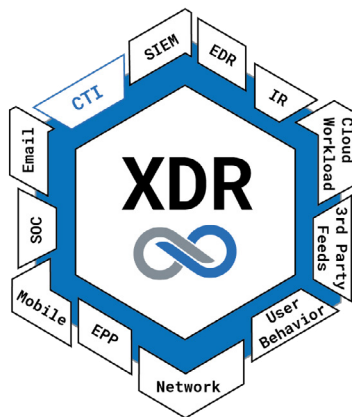


Figure 1 – Topological view of XDR

So why does XDR matter in the context of this book? Well, the automated ingestion and correlation of intelligence data helps to decrease the burden of “alert fatigue” for SOC analysts and incident responders. By providing more contextual information concerning incidents and alerts, incident responders are better informed to react swiftly and decisively. In addition, the data can be used for the automation of IR playbooks, to help orchestrate workflows and processes during incidents.

How then, do you produce, correlate, and consume CTI to empower XDR enabled solutions and services?

As prevention is always better than a cure, the ultimate solution needs to be more proactive than reactive. The hunted must become the hunter, and for this pursuit, Cobalt Strike Team Servers our quarry.

SO, YOU WANT TO GATHER CYBER THREAT INTELLIGENCE?

WHAT WILL YOU FIND IN THIS BOOK?

In this book, the BlackBerry Research & Intelligence Team presents a system for hunting the Internet for instances of Cobalt Strike Team Server, which is the command-and-control (C2) server for one of the most pervasive threats deployed by modern threat groups: Cobalt Strike Beacon.

In addition, we present our Cyber Threat Intelligence (CTI) lifecycle, which outlines our multi-phase approach to building intelligence-led protection for products and services underpinning most XDR products and services. The lifecycle outlines the following phases:

- Project planning and direction
- Data collection, processing, analysis, and dissemination
- Continuous improvements via evaluation and feedback

By following our CTI lifecycle to hunt for Team Servers, and extracting configurations from the Beacon payloads they serve, we aim to demonstrate how you can leverage the resulting dataset to provide powerful intelligence insights. These insights can help to reveal clusters of servers associated with known threat groups and campaigns, as well as links between them that were previously unseen, empowering you to expose correlations between seemingly disparate network infrastructure.

Finally, the resulting intelligence can also be leveraged to provide actionable Indicators of Compromise (IOCs) to all XDR stakeholders, including defenders, hunters, analysts, and investigators alike. These will help you to:

- Defend your organization
- Produce in-depth CTI reports
- Better understand the threat landscape
- Stay ahead of the curve – give better advice to your C-level executives and security teams so that they can make well informed security-oriented decisions

WHO IS THIS BOOK FOR?

This book is for anyone with an interest in gathering Cyber Threat Intelligence, those who want to further their understanding of Cobalt Strike, or those who simply enjoy a good technical read.

That said, the people who might derive the most reward from this book may include:

- Threat Intelligence Analysts
- Threat Hunters
- Incident Responders
- Forensic Investigators
- SOC Analysts
- Red Teamers

HOW CAN YOU BENEFIT?

By defining a CTI lifecycle, we present a blueprint to help you with creating one that fits your own needs. It can also be used to help build your own automation platform for harvesting and disseminating cyber threat intelligence.

Walking you through our lifecycle, we begin the collection phase by hunting for active Cobalt Strike Team Servers. Once the true cyber adversaries' Team Server instances are identified, it gives us a unique opportunity to reveal trends and patterns within the data. These insights can help to perform a variety of useful things, such as:

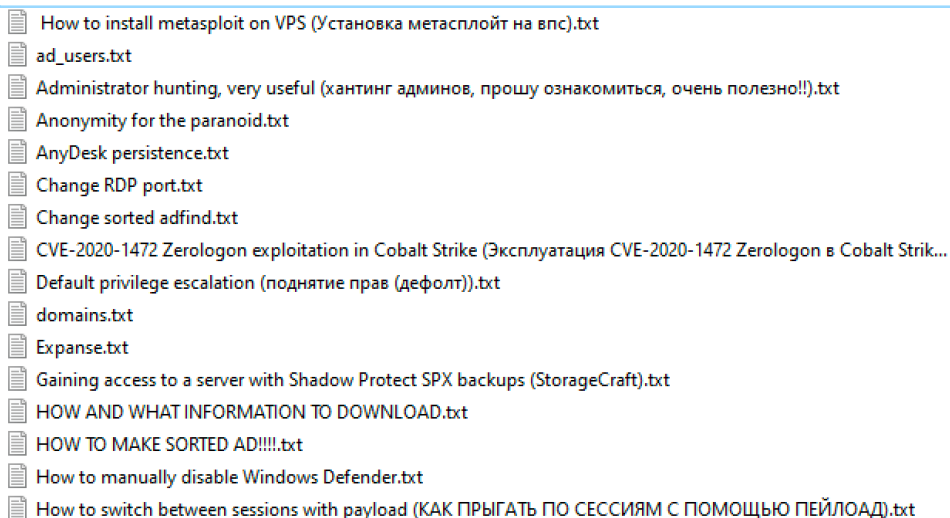
- Building profiles of threat actors
- Broadening knowledge of existing threat groups
- Tracking both ongoing and new threat actor campaigns
- Providing actionable intelligence to SOC analysts and IR teams
- Fine tuning security products and services under the XDR umbrella

While we use the example of Cobalt Strike in this book, we hope this exercise sparks your imagination and inspires you to use this for other threat intelligence quests. This industry thrives because it is populated by so many individuals who are passionate about the sharing of information, including tools, tips and techniques. By adding our contribution, we hope to keep this altruistic tradition alive.

All these things aside, we hope that in reading this book you may learn a thing or two, have a laugh along way, or even gain insight into our processes for the purposes of competitive intelligence!

WHY ARE WE WRITING ABOUT IT NOW?

The unfortunate reality is that the rate of cyber intrusions has grown exponentially in recent years, with high-profile ransomware attacks becoming a staple feature of the daily news cycle. The ease with which threat actors can arm themselves with advanced adversarial tooling means that what was once quite a complex affair is now nearly effortless. In some cases, it is as simple as copying and pasting a few commands and pressing a few buttons, as we saw with the leaked Conti ransomware playbook.



- How to install metasploit on VPS (Установка метаспloit на вps).txt
- ad_users.txt
- Administrator hunting, very useful (хантинг админов, прошу ознакомиться, очень полезно!).txt
- Anonymity for the paranoid.txt
- AnyDesk persistence.txt
- Change RDP port.txt
- Change sorted adfind.txt
- CVE-2020-1472 Zerologon exploitation in Cobalt Strike (Эксплуатация CVE-2020-1472 Zerologon в Cobalt Strik...
- Default privilege escalation (поднятие прав (дефолт)).txt
- domains.txt
- Expanse.txt
- Gaining access to a server with Shadow Protect SPX backups (StorageCraft).txt
- HOW AND WHAT INFORMATION TO DOWNLOAD.txt
- HOW TO MAKE SORTED AD!!!!.txt
- How to manually disable Windows Defender.txt
- How to switch between sessions with payload (КАК ПРЫГАТЬ ПО СЕССИЯМ С ПОМОЩЬЮ ПЕЙЛОАД).txt

Figure 2 – Truncated list of Conti ransomware playbook files – Translated

BleepingComputer

Translated Conti ransomware playbook gives insight into attacks

<https://www.bleepingcomputer.com/news/security/translated-conti-ransomware-playbook-gives-insight-into-attacks/>

While not the only culprit, Cobalt Strike Beacon has been the common denominator in these attacks time and again. Lesser-financed and lesser-resourced groups – as well as those just looking to blend in with the crowd – need look no further than cracked, leaked or trial versions of Cobalt Strike. The low barrier to entry this provides, with the ease of propagation through botnets and other distribution services, has acted as a catalyst for the ransomware epidemic and expedited its rate of growth.

To improve our own intelligence-led protection and correlation of malicious instances of these components, BlackBerry created an automated system to scan for Cobalt Strike Team Servers. It downloads Beacons then extracts and stores their configurations for further processing, analysis, and dissemination.

The aim of this book is to aid the security community by sharing this knowledge, presenting the steps we've taken to create this automated system, and most importantly, demonstrating how to derive meaningful threat intelligence from the resulting dataset. This information can then be used to provide insights, trends and intelligence on threat groups and campaigns.

HOW IS THIS BOOK ORGANIZED?

This book is organized into six chapters. It begins with an introduction to our CTI lifecycle, where we outline our processes and methodologies. Next, we'll delve into the specifics of how to develop a system to perform automated hunting of Cobalt Strike Team Servers, which can yield useful and meaningful intelligence data.

We will then introduce Cobalt Strike Beacon and its configuration profiles, as well as a full table of configuration options and common values for quick reference. We will use the resulting knowledge and dataset to dig deeper into insights and identify trends, uncovering some unexpected revelations along the way.

Finally, we will enrich our dataset with open-source intelligence (OSINT) using our Threat Intelligence Platform (TIP), and look to uncover new correlations and groupings, before circling back to reassess our CTI objectives and findings in a debrief.

DISCLAIMER

It is worth mentioning that no Team Servers were harmed in the making of this book! All findings throughout the book originate from Beacon analysis. We have not installed, remotely or locally operated, reverse engineered nor debugged Team Server to arrive at any of our findings or conclusions.

CHAPTER ONE

BEYOND THE HYPE



CYBER THREAT INTELLIGENCE

The most visible aspects that many people associate with CTI are the cool names and awesome logos given to vulnerabilities and threat actors such as HeartBleed, Shellshock, OceanLotus, and WizardSpider. More than that, CTI is a discipline, albeit one in its infancy. And as such, it needs a little formalizing.

Recent advances in cybersecurity technology, such as XDR, certainly necessitates the need for more formal and mature processes. CTI is now widely used to underpin XDR solutions. This means that intelligence insights are leveraged to enhance protection and intelligence correlation, leading to an increased efficacy for security products and a reduction in alert fatigue for SOCs. We call this intelligence-led protection and correlation.

The current CTI landscape draws from multiple sources, including the military, intelligence agencies, universities, and the private sector, to name a few. All these influences have offered significant improvements to what was once simply termed “threat research”, and have helped to evolve CTI processes, workflows, and paradigms in a short period of time. The speed of development in this area inevitably leads to some confusion and a lack of consensus on the right way to approach CTI creation.

We’d love to say we have the silver-bullet solution for how to do it properly. In reality, this book aims to highlight some of the common phases and most critical areas so that we are all on the same page (no pun intended). When you get hooked on CTI and want to improve your organization’s program, there are numerous resources, such as books, papers, talks, blogs, and training programs that can help you.

Understanding CTI as a lifecycle – where people, processes, and technology work in harmony – will lead to the production of intelligence that can be used to assess risk, identify threats, and make informed decisions. And if you insist, you can even give that intelligence product a cool name and a supervillain-esque personification.

Now that we’ve introduced the concept of CTI, almost everyone will have a different interpretation of what that means. To avoid any misunderstandings, here is our working definition of CTI that will help you to understand what we are all trying to achieve...

“Cyber Threat Intelligence collects information to answer specific questions (such as who, what, where, when, how, or why) about a person or thing that is likely to cause damage or danger to computers or networked systems.”

That’s kind of wordy, so feel free to take the sentiment and create your own definition for your organization. It’s important to have a well-understood definition that works for you.

Having both your team and management coalesce around a well-formed idea is hugely beneficial. It keeps everyone focused on achieving their goals, while management is clear on the outcomes the team will deliver.

Having defined our deliverables, let’s put some thought into how to do it. While there is a great deal of creative thinking involved in CTI, it should not be the sole requirement of the team. Without a defined framework, creative thinking can spark inspiration, but it will have no way of following through on its promise.

By agreeing on a framework, and then developing the people, processes, and tooling to support its execution, team members will be able to understand their responsibilities on a tactical and strategic level. The ideal situation is to have well-trained people follow a repeatable process that is supported by the appropriate tooling, which aligns with a scientific methodology. If you achieve this, it will enhance (rather than rely on) the intuition of individuals.

OUR CTI LIFECYCLE FOR AN XDR WORLD

There are many versions of the CTI lifecycle. The one we choose to use is adapted from multiple lifecycles and allows us to build repeatable, iterative processes to support the production of intelligence that works for all stakeholders in our organization.

This section is not meant to be the definitive CTI lifecycle. We hope that it will spawn ideas that you can assess for your own organization and help inform the lifecycle you choose to follow. It also serves as scaffolding for further information laid out in this book.

Each of the processes, scripts and analysis descriptions discussed in this book can be tied back to a distinct phase of the lifecycle. Thinking of it in this way can provide order to what might seem to be chaos. (If not chaos, then maybe Thanos - chaos' older, more-chaotic sibling.)

The lifecycle we describe in this section isn't prescriptive in terms of the processes or technology required. That's all up to you to decide. The framework we're providing allows you to decide the appropriate people, processes and technology that work best, to accomplish the goals of each phase.

Likewise, there is no set number of processes to include in each of the phases. It all depends on what is needed to achieve your intelligence requirements.



Figure 3 –Our CTI lifecycle

Speaking of intelligence requirements, this leads us nicely onto the first (seemingly most mundane, certainly most overlooked, and yet critically important) phase.

PLANNING & DIRECTION

During the planning and direction phase of the CTI lifecycle, we like to set up two key components:

- The question (or questions) to be answered
- The audience who requires the answer(s)

That sounds easy, right? That's why this phase is often overlooked or poorly thought through. But the results of paying mere lip service to planning and direction will haunt you. Getting this phase right will lead to greater efficiency and focus for your team, as well as a better intelligence product for your audience. This is the best chance to rid yourselves of ambiguity and assumptions about what you are trying to achieve.

Thinking back to the description of CTI, the statement, 'collects information to answer specific questions' stands out. The planning and direction phase is where you define the question you are going to answer throughout the rest of the lifecycle. Everything you do should be focused on answering the question defined in this phase.

While we are talking about questions; not all questions are created equal when it comes to intelligence requests. Questions that are generally narrow in scope and those which form a closed loop work better. They help us by creating tailored knowledge to support specific decisions the individual or group is looking to make.

Understanding the difference between broad and closed-loop questions can be a little tricky, so let's look at some example questions:

What is the biggest cyber threat today?

This question is too broad. Simply put, there are too many different ways to answer it. People can reasonably have different interpretations of what is required. For example: How do you define 'cyber threat?' How do you define 'biggest?' Who is the target that you're most concerned with? As there is no focus around what is required, the resulting product will not be useful in supporting any meaningful decision. This might feel satisfying to explore, but for practical purposes, it will be wasted effort.

A more closed-loop example might be:

What public-facing vulnerabilities have been most commonly exploited in the last three months?

This question is very specific. It clearly delineates what threat we're concerned with (public-facing vulnerabilities), what aspect of it is most relevant (most commonly exploited), and it gives a limited time-frame (three months).

While you don't necessarily need to have this much specificity, the more information you can include in the question, the better your answer will be. This kind of question will result in the team understanding what is required of them, which means less chance of researchers going off-track. The answer will lead to actionable intelligence and may still be quite satisfying to explore.

Having this narrow focus when producing the intelligence product is key to keeping your team on track to produce what is needed. This is where things can get a little nuanced; during the process of answering the question, the team is going to have to work with a lot of data. Some proportion of this data may not be relevant to the question being set. But if your team has taken the time to collect, process and analyze this data, don't waste that work (you might just get a book out of it!).

In the world of XDR, this data has its place and should not be discarded as waste. To illustrate the point; while refining sugar cane into sugar, one by-product of the process is molasses. Where sugar is the sweet and shimmering answer to the question, the mineral-rich molasses is the analyzed data that is irrelevant to the desired outcome.

Other teams will be able to make use of this gooey goodness, and they can make something truly valuable out of it. Spread the love and find the teams in your organization that can make the best use of the results of your hard work.

In this phase you should also think about exactly who is going to consume the produced intelligence, and what their requirements are. It is all too easy for a passionate researcher to get caught up in a cool exploit or innovative obfuscation technique. But if the intended audience is a CISO looking to decide what tooling to buy based on trends in attacker tactics, techniques, and procedures (TTPs), an intelligence product that goes down a different rabbit hole is worthless for answering this question.

An audience might think differently than you, and they could require things that you would disregard. One way to help define who needs which information is to describe three different types of intelligence:

- **Strategic** - Broader trends for a non-technical audience
- **Tactical** - Outlining TTPs of threat actors for a technical audience
- **Operational** - Details about malware and attacks for a technical audience

Hopefully, you now understand the importance of this phase and can see how putting a little more time and thought into it will pay off for the rest of your endeavors.

COLLECTION

Now that we know what facts we seek and who will be consuming those facts, we need some data to work with. Before you get all excited and grab all the data from All-The-Things™, keep in mind the question that we are trying to answer. We need data to answer that question. Specifically, we need relevant data.

Collection is where we gather that relevant data. This could be both internal and external data, including:

- File hashes
- IP addresses
- Domains
- Political news articles
- Historical information
- Logs
- IR reports
- Blogs
- Dark web
- Social media sources
- Code snippets from online repos
- Data dumps

From a very generic perspective, external data sources are usually easier to access and consume because they come from products that are made to be used that way. External sources typically have a well-defined interface for extracting data, such as via API, or export functionality within the user interface.

Internal data sources require more time and development effort to introduce because they usually aren't coming from products designed with that functionality in mind. Pulling that data out might require extra processes from teams like IR or the SOC that are busy with their other daily responsibilities. It might also require further development of internal tools, diverting development resources away from improvements to the primary function of the tool is a tricky balancing act.

The trade-off to consider is that while internal sources contain information that is way more relevant to your organization, an over-reliance on external sources might not give you the insight you require.

Regardless of where you get your data from, collection is the perfect place to introduce automation. As you read through the rest of the book, look at the queries and processes used to automate the harvest of Cobalt Strike Beacons. You should see that they can all be performed by either a human or in an automated fashion. Where things can be automated, try to make that a reality. The biggest benefit of having humans in the mix will become apparent soon.

PROCESSING

Now you have data, but it might not be ready yet for human consumption. Processing is where you manipulate the data. You organize it, label it, translate it, deobfuscate it, decrypt it, and filter it. You make it ready for the analyst to use. As with the previous phase, automation is pretty much a pre-requisite for the processing phase. The number of manipulations you are likely to have to do, over the sheer volume of data you will inevitably gather, is a huge waste of your most precious resource – your team. Not to mention, this sort of processing is soul-destroying drudge work.

The final thing to think about as you're processing data is how to provide a curated dataset somewhere that your analysts can interrogate it. You can make this as complicated or simple as you like. Excel pivot tables can be a pretty powerful starting point. Maltego and Jupyter Notebook offer more advanced visualizations. And for the truly adventurous, PyQt5 (Figure 4) makes custom data visualizations very easy.

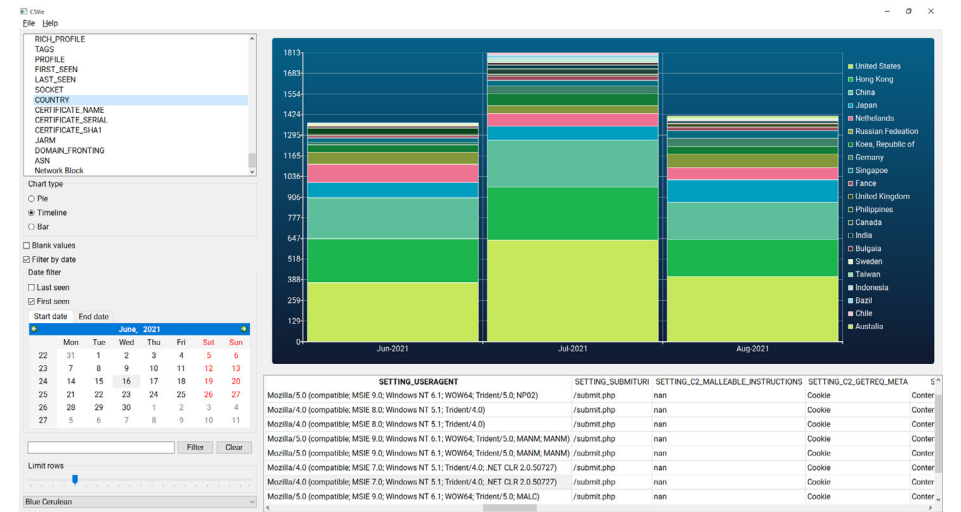


Figure 4 – The PyQt5 based visualization tool we developed for viewing our Beacon datasets

ANALYSIS

So now we have a question, we have an audience, and we have relevant data. This is the point at which humans cannot be replaced. In a phase shrouded by the psychology of cognitive biases, competing hypotheses, and a myriad of reasoning techniques, we attempt to answer the initial question we were assigned.

When conducting analysis, it is important to keep the two outputs from planning and direction clear in your mind: what is the question, and who is it for?

If you are in the intelligence-creation space, you are (or at least should be) a curious person. While this is a terrific quality for analysts, it has a significant downside. You always want to know more, so you might struggle with where to stop analyzing.

On the surface, understanding more about any given subject is better, right? Creating a masterpiece of a report that takes four months to write means the data you reference might be out of date and therefore not actionable. Conversely, if you work quicker and get the data out sooner, you might not have the time you need to assure the accuracy of the data. This balance between time and accuracy versus completeness is something everyone in the field battles with.

To help with this balance, let us go back to planning and direction. Who is going to consume the intelligence, and what do they need from it?

As a very rough, very generic guide, we can look back at the different types of intelligence:

- **Strategic** - Greater need for accuracy and completeness, less time sensitive.
- **Tactical** - The middle ground. As complete as it can be, while being delivered as quickly as possible.
- **Operational** - Needs information as close to real time as possible; some lack of accuracy is tolerated.

Before we move on to the next phase, it is worth noting that entire careers have gone into understanding how to analyze data. There is no way we can hope to do the science of research and intelligence analysis any justice in this book. If you are wanting to understand the way people think and reason, Richard J. Heuer's book *Psychology of Intelligence Analysis* is a great place to start.

DISSEMINATION

Once you break away from the fun stuff, it is time to gently place your intelligence baby into the hands of its new owner(s). Your creation needs to be released for consumption by vested stakeholders. In reality, this will likely involve many teams and individuals who are involved in providing XDR services.

Referring to the initial phase of planning and development again, the medium for this publication will depend on the audience and their requirements. As with everything contained in this section, there is no right or wrong way, but there are factors to consider with the different types of intelligence products available.

Security Operation Center (SOC) analysts and Incident Response (IR) teams are going to want an intelligence product they can parse quickly, and perhaps load into tooling. Executives are going to require something that is easily understood, preferably in report format, or potentially as a briefing with slides and key findings.

Remember where we said that the planning and direction phase will haunt you, if it's done sloppily? If you give your IR team a 40-page PDF file, or your executives a list of contextualized IOCs, people aren't going to see the value in the intelligence you have lovingly crafted. Delivery of the information should not be an afterthought.

There it is: we've planned, collected, processed, analyzed the data. And now we've delivered it to the stakeholder. We're done, finished, time for the next... But wait! We're not quite there yet. There's one last, equally important, phase to consider before we can call our lifecycle complete.

EVALUATION AND FEEDBACK

We made it to the final phase, and it's time to evaluate the delivered item against the goal we created in the planning phase. Did we deliver what we wanted to? Was it accurate? Was it timely?

We can't possibly attempt to answer those questions without something to compare it to. This once again highlights the importance of the planning and direction phase.

Aside from evaluating the product, we should highlight any deficits that were discovered in any phase of the lifecycle so that improvements can be made. Because this is a cyclical process, if this step is missed, it will mean a degradation of the service over time, and repeated failures in future projects.

Evaluating the lifecycle can help us with automation too. Full automation is not always immediately achievable. It often requires an iterative approach over time, with cyclical analysis and development driven by insights gleaned from repeated manual processing, as well as feedback from stakeholders.

An approach that we've implemented, which we've found helps us when analyzing the success and failures of each phase, is to look at each one through the lens of the three main components needed to deliver it:

- People
- Process
- Technology

For example, when considering the collection phase, ask the following questions:

- Do we have the skill set we needed within the team to identify the relevant data required?
- Does the process for gathering the data execute in a timely fashion?
- Do we have the right tools in place to ingest the data?

If you look at it from this perspective, you can then make recommendations and secure funding with specificity. And if you want to go the extra mile, you can quantify the improvements to make a business case.

ALIGNING THE STARS: THE CTI LIFECYCLE

OK, now that we've gotten all the administrative details out of the way, let the games begin.

When you use the CTI lifecycle within your organization, the points at which the requestor or customer will interact with the lifecycle are limited. They will be involved in the planning and direction phase, helping to define what they need and how they need it. The next point at which they will be involved is when they receive the product, in the dissemination.

The majority of the work you do will be completed out of the spotlight. For the purposes of this book, we will walk through those phases, to give inspiration in how you can approach the lifecycle within your organization.

Quick quiz: what is the first phase of the lifecycle? You got it, Planning & Direction!

So, for the purposes of this book – what was our question, and who was our initial audience?

Eric Milam, our Vice President of Research and Intelligence, tasked the Research & Intelligence Team with providing intelligence to all XDR stakeholders to help them proactively protect and defend against Cobalt Strike.

We then asked ourselves....

"How do we proactively defend against Cobalt Strike?"

There is a lot in that question. If you look at it through the advice above, does it meet the requirements of a narrow question?

Not really! It is broad, and it isn't an intelligence question. But it has an intelligence component.

So, we worked on what was actually required from the intelligence side of the team. In order to answer the bigger question, we have several teams who need to consume our intelligence, including SOC teams, product engineering, data scientists and analysts, all contributing to products and services under the XDR umbrella.

That one question posed by Eric became several questions, with different audiences across the XDR solution space, and therefore different deliverables.

Our SOC require contextualized alert information, and asked:

"How can we improve incident correlation and reduce alert fatigue?"

Product engineering wants a better understanding of the operation of Cobalt Strike Beacon, posing the question:

"How can we fine-tune EDR to detect Beacon payloads?"

Data scientists want labelled data for training models, wondering:

"What features are helpful for training models to classify Cobalt Strike Beacon payloads and configurations?"

IR want intelligence correlation, IOCs and TTPs, asking:

"How can we improve correlation and campaign tracking relating to Cobalt Strike?"

Finally, intelligence analysts asked:

"How can we track Team Servers and campaigns?"

Throughout the remainder of this book, we'll demonstrate our CTI lifecycle by building an automation system to collect and process Cobalt Strike Beacon payloads, uncovering over 6,000 Team Servers along the way. We'll provide our insights and trends from analyzing over 48,000 Beacons served from those 6,000+ Team Servers, and we also exhibit how intelligence correlation can be performed to enhance our knowledge of threat groups.

Finally, we will debrief, and assess how our results helped answer the questions posed by the various stakeholders.

CHAPTER TWO

ALL YOUR BEACONS ARE BELONG TO US



To defend against Cobalt Strike, we must first understand how it operates. Cobalt Strike works in an agent and server configuration. Each Beacon is deployed (usually surreptitiously) as an agent on the endpoint and is configured to communicate with a Team Server that acts as the C2.

One is rendered useless without the other, which gives us a couple of options in terms of hunting and detection capabilities. We can choose to detect and respond to either the Beacon or the Team Server, however, there are reasons why you may choose one over the other.

Detecting and responding to the Beacon likely means that a threat actor is already active on our networks, or that there has been a patient zero victim. This approach is therefore largely reactive. Detecting the Team Server has no such requirement and means we do not have to wait for a device to be targeted before taking action to defend ourselves.

To be proactive, which is the ideal scenario, we must be actively looking to locate and identify Cobalt Strike Team Servers in the wild. Ideally this would happen as soon as possible once a new Team Server is deployed. This would allow us to take preventative actions, thereby cutting the head off the snake before it ever has a chance to get close enough to bite.

So, where can we look to find a source of Team Servers for our data collection purposes?

DATA COLLECTION

DEFINING THE SCOPE

There are several data sources we can use to generate a list of Cobalt Strike servers that we will want to defend against. These sources can include the following:

- Threat intelligence feeds
- Industry reports
- Incident response data
- EDR alerts
- Threat hunting

While they are still valuable, many of these sources are reactive in nature and place defenders on the back foot. By the time something ends up in an intelligence report or has triggered alerts in your SIEM, something bad has potentially already happened.

There are several public methodologies for identifying Cobalt Strike Team Servers or active Beacons. These can include, but are not limited to:

- Default security certificates
- Default port (50050/TCP)
- A tell-tale extra null byte in HTTP server responses
- Jitter and sleep interval analysis
- JARM signatures
- DNS redirector response

RecordedFuture

A Multi-Method Approach to Identifying Rogue Cobalt Strike Servers

<https://www.recordedfuture.com/cobalt-strike-servers/>

As already stated, our aim is to stay one step ahead of the bad guys and detect Team Servers in the wild. To this end, we have three main options:

- Scan the entire internet using a custom-built scanner with the purpose of detecting and analyzing Cobalt Strike Team Servers
- Leverage well-known and established scanning services already available on the internet such as Shodan, Rapid7, Censys or ZoomEye
- Build a hybrid system that leverages public services in conjunction with a private, more targeted scanner

All options have their strengths and weaknesses. They require differing levels of investment and have different barriers to entry for any organization looking to implement such a system.

Building and operating a bespoke Internet-wide scanner and analyzer is the best option in terms of the potential quantity of results. It also offers the best ability to add customizations. But this is also the most expensive option in terms of the time and skills required to implement it. It might be beyond many organizations' capabilities or budget. The use of public scanning services can be helpful for organizations that do not have an existing way of discovering or tracking Cobalt Strike infrastructure.

However, without an additional layer of human or automated analysis for quality assurance, these services might not yield optimal results. You could not achieve a high level of certainty that a server is indeed hosting a Cobalt Strike instance.

Building a hybrid system is a happy medium between these two approaches. This should provide results that have a high level of certainty, but in a more cost-effective manner. Granted, you might not have the same volume of results as from a bespoke system, but it would certainly still offer a good return on investment.

CRAFTING SOME QUERIES

Trying to build a scanner to scan the entire Internet, to accurately fingerprint the systems found, and then to store all the resulting data is no mean feat. Don't forget to add to this the potentially significant effort required to procure the budget for your AWS (Amazon Web Services) bill if you want to do this continuously and rapidly, and to store the results for any length of time.

This is where services like Shodan can make life easier, as they have already done the leg work for you. Other researchers have used similar services like Censys and ZoomEye. Or you can opt to use datasets from Rapid7 instead for Cobalt Strike hunting.

For this paper we will focus on Shodan, but Rapid7 Open Data was also invaluable in our data collection phase. You may decide to use one or even all the services mentioned.

Shodan

<https://www.shodan.io/>

Rapid7 Open Data

<https://opendata.rapid7.com/>

Censys

<https://censys.io/>

ZoomEye

<https://www.zoomeye.org/>

Identifying Cobalt Strike team servers in the wild by using ZoomEye

<https://80vul.medium.com/identifying-cobalt-strike-team-servers-in-the-wild-by-using-zoomeye-debf995b6798>

The important part of this phase is getting relevant data to feed to the next stage of our analysis. Firstly, we need to craft search queries to unlock the value in Shodan's data. To limit false positives, these queries need to be based on known Cobalt Strike Team Server characteristics. The results of these queries will still need further scrutiny and processing to increase the level of certainty that a server is hosting Cobalt Strike.

It will take time, experimentation, and regular updates to craft a good set of queries that can account for the different versions of Team Server. These queries should also include instances where the threat actor has customized their deployment, causing it to go undetected by an existing query set.

Here, we have crafted a query that can be used to detect the '404 Not Found' HTTP response returned from the NanoHTTPD server used in the backend of a Cobalt Strike Team Server.

```
"Content-Length: 0" AND "HTTP/1.1 404 Not Found" AND "Content-Type: text/plain" AND "Date:"
```

Figure 5 - Shodan query to detect Cobalt Strike '404 NOT FOUND' response

This query searches for a HTTP server returning a '404 Not Found' response that has a content length of zero, a content type of 'text/plain,' and which returns a 'Date' header.

```
HTTP/1.1 404 Not Found
Date: Fri, 24 Sep 2021 15:08:20 GMT
Content-Type: text/plain
Content-Length: 0
```

Figure 6 – Default NanoHTTPD HTTP header response

The number of results returned via this Shodan query is huge, with more than 322,000 in total. The majority of these would likely be false positives. This is due to the way Shodan queries operate; they will trigger on any systems that contain the values specified in our query, including systems that contain other headers in addition to those specified.

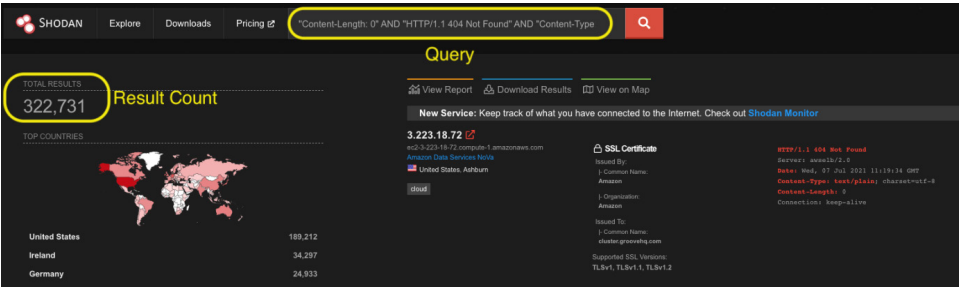


Figure 7 – Shodan Query results for "404 NOT FOUND" response from Cobalt Strike Team Server

For those familiar with programming logic or string comparisons, it is best to think of Shodan queries as a 'contains' comparison, rather than 'equals'. To work around this, we will require a tighter, more specific query to filter some of these extraneous results out.

For example, if we wanted to remove results that contain a 'Connection' header, we can append 'AND NOT "Connection:"' to our existing query. This would significantly reduce the number of results and cut down on false positives.

The alternative to filtering at the query level is to perform some additional processing of the results, using automation or scripting.

Depending on your level of Shodan API access, you might be forced to refine the query quite a bit, or else you risk exceeding your API key limitations. You will definitely need more than one query to cover the range of Cobalt Strike server configurations or customizations, so make sure you adjust your approach to suit your API limits.

There needs to be a balance between having a query that is not so tight that it creates false negatives, but also not so loose that you create false positives (which in effect are wasted API query results). API limitations aside, in most scenarios a false positive is more favorable than a false negative. False positives can be whittled down later, but false negatives are missed Team Servers, which are potentially active in-the-wild and used to conduct attacks.

Another query that has provided valuable results for detecting Cobalt Strike Team Servers is based on JARM fingerprinting. JARM is a Transport Layer Security (TLS) fingerprinting tool developed by Salesforce, which they have leveraged to detect Cobalt Strike Team Servers and other malicious servers.

Salesforce

Easily Identify Malicious Servers on the Internet with JARM

<https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a>

Shodan added JARM search functionality in November 2020, and it is proving to be a powerful tool in the threat hunter's arsenal. The Cobalt Strike Team Server is written in Java, and each Java TLS stack has a very specific JARM fingerprint. As Java 11 is frequently used as the build version for a great number of Cobalt Strike Team Servers, its JARM fingerprint has also become a JARM fingerprint for Cobalt Strike.

```
ssl.jarm:"07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1"
```

Figure 8 - Shodan Query for Cobalt Strike/Java 11 SSL JARM fingerprint

Cobalt Strike

A Red Teamer Plays with JARM

<https://blog.cobaltstrike.com/2020/12/08/a-red-teamer-plays-with-jarm/>

SANS

Threat Hunting with JARM

<https://isc.sans.edu/forums/diary/Threat+Hunting+with+JARM/26832/>

Searching a known Java 11 JARM associated with Cobalt Strike, we received a little over 6,000 results:

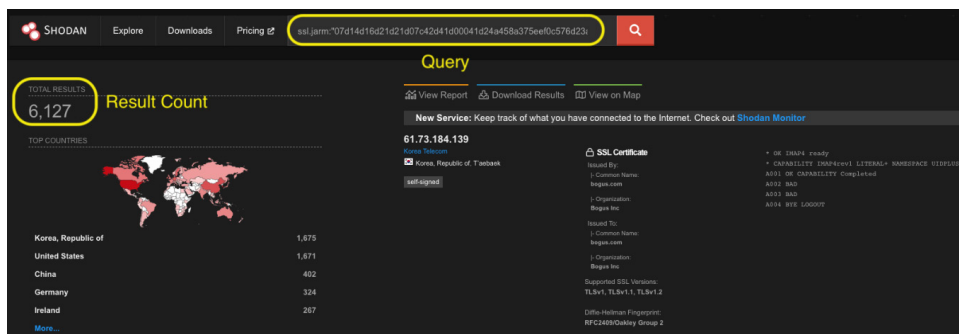


Figure 9 - Shodan Query Results for SSL JARM fingerprint for Cobalt Strike Team Server

These results contain Cobalt Strike Team Servers as well as legitimate servers running the Java 11 TLS stack, so false positives will be present.

We also need to consider that spoofing JARM signatures is a possibility, whereby a server can be configured to masquerade as a Cobalt Strike Team Server from a TLS/JARM perspective. These spoofed servers might be configured to act as a honeypot that could be used to detect systems like ours, that are attempting to discover Cobalt Strike Team Servers in the wild. These servers can then be blocked during future deployments, potentially thwarting our scanning efforts.

Stefan Grimminck

Spoofing JARM signatures. I am the Cobalt Strike server now!

<https://grimminck.medium.com/spoofing-jarm-signatures-i-am-the-cobalt-strike-server-now-a27bd549fc6b>

Additionally, threat actors with an awareness of JARM fingerprinting could also modify their TLS stack in a way that alters their JARM fingerprint to evade detection. Looking at the top 10 JARM fingerprints for Team Servers we have observed in the wild we can confirm that this is the case. The top result by far is the JARM fingerprint for the Java 11 TLS stack, but there are several notable deviations from this.

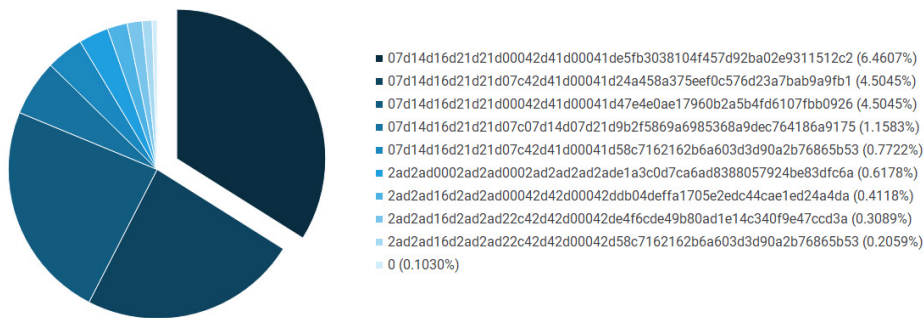


Figure 10 – Top 10 JARM fingerprints

Other queries can be based on criteria such as Cobalt Strike’s default, self-signed SSL certificate, which we’ll take a closer look at later in the book.

```
ssl.cert.serial:146473198
```

Figure 11 - Shodan query to check for Default SSL Serial

This SSL certificate should ideally be changed from the default prior to a live operation or engagement, but people often neglect to change it when they deploy a Team Server. This gives us an opportunity to discover servers that still use this certificate, whether intentionally or not.

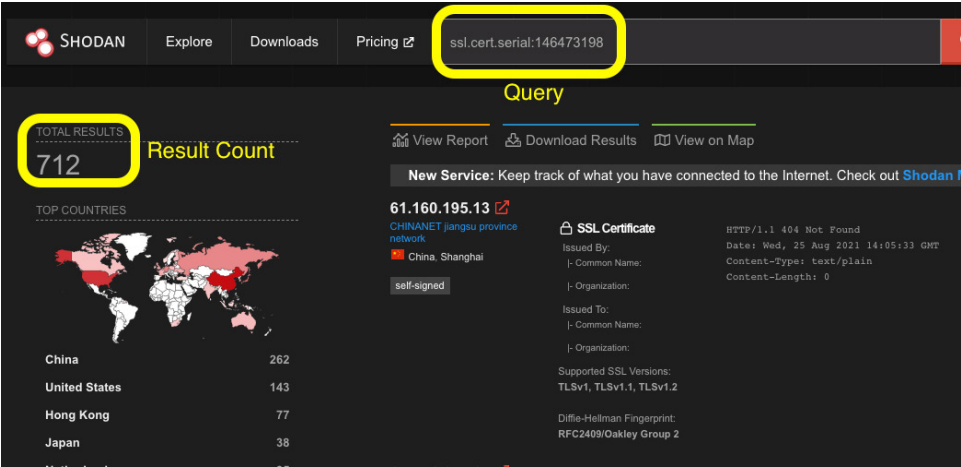


Figure 12 - Shodan Query Results for SSL certificate serial for Cobalt Strike Team Server

Lastly, we can also craft queries based on the numerous malleable C2 profiles for additional coverage (we'll cover these profiles in more detail in chapter 3: We are Beacon).

One such example would be a query to detect Team Servers using the Microsoft Update malleable C2 profile. This profile is configured to use a certificate common name of *www.windowsupdate.com*, to masquerade as a legitimate Microsoft certificate.

```
1 #
2 # Microsoft Update
3 #
4 # Author: @bluscreenofjeff
5 #
6
7 #set https cert info
8 #information assumed based on other Microsoft certs
9 https-certificate {
10   set CN      "www.windowsupdate.com"; #Common Name
11   set O       "Microsoft Corporation"; #Organization Name
12   set C       "US"; #Country
13   set L       "Redmond"; #Locality
14   set OU      "Microsoft IT"; #Organizational Unit Name
15   set ST      "WA"; #State or Province
16   set validity "365"; #Number of days the cert is valid for
17 }
18
```

Figure 13 – HTTP certificate from Microsoft Update malleable C2 profile

When we query for servers using this certificate common name, we get far fewer results when compared to the other queries. But this time we get a higher likelihood of true positives.

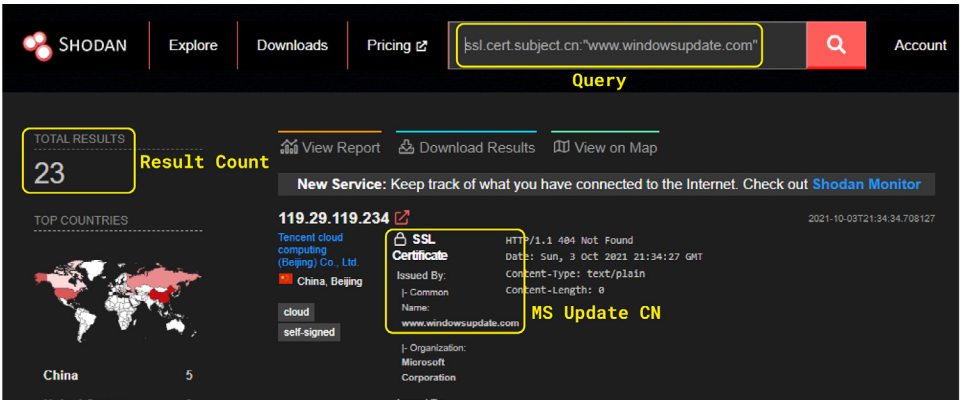


Figure 14 – Shodan query results for “Microsoft Update” HTTP certificate

With some time and experimentation, and further knowledge of malleable C2 profiles, common certificates, and HTTP response headers, we can craft multiple queries that will return an abundance of data we will need to further validate for potential Team Server activity.

DATA VALIDATION

Now that we have crafted some queries and started to gather data, how do we validate the results?

There are a few questions that we can ask of the data, which can increase our confidence of a valid detection:

1. Do any servers appear in multiple result sets? i.e. Are there any detection overlaps?
2. Have any of the collected servers been reported via OSINT channels, threat intelligence feeds, or been observed attacking other organizations?
3. Can we coerce the Team Server to serve up a Beacon?

At a minimum, we need to interrogate the datasets and intelligence feeds for detection overlaps and duplications and check if any of the servers have already been observed carrying out malicious activities. Detection overlaps can be used to our advantage here; if we have two or more differing queries or datasets that return data containing the same IP address, then it can increase the likelihood that the discovered IP is in fact an operating Team Server.

The ideal scenario is one where we can force a server from our data set to serve us a Beacon. If we can do so, then we will know for certain that we have discovered a Team Server and can mark it for the next stages of our CTI lifecycle.

This will not always be possible, as Team Servers can be protected behind redirectors that can limit connections to the Team Server itself, thereby preventing us from retrieving a Beacon. In this instance, there are further ways of detecting Cobalt Strike redirectors that can be added into our validation process to further increase our detection confidence.

Sticking with the objective of retrieving a Beacon from a Team Server, we need to know how we can emulate a valid stager check-in so that we are served a Beacon. There is a bit of behavior that was implemented purposefully to allow interoperability between Cobalt Strike and Metasploit Framework generated stagers that can help us here. Both Metasploit and Cobalt Strike stage their payloads in such a way that a specially crafted HTTP request – where the Uniform Resource Identifier (URI) matches a specific checksum8 value – will cause the Team Server to serve a Beacon.

Depending on the mode of operation and the architecture of the payload being staged, the URI may need to be of a specific length and have a specific checksum8 value. The breakdown of how this works for Cobalt Strike can be seen in the table below.

Architecture	Mode	Checksum8	URI Length	Example URI
x86	Normal	92	ANY	/aaaaaw
x64	Normal	93	4	/dVbA
x84	Strict	92	5	/aa910
x64	Strict	93	5	/ab820

Table 1 - Stager URI breakdown

Armed with this knowledge, we can develop a simple Python script that can generate a URI string to satisfy any checksum8 validation and URI length checks. This script could therefore ultimately be used to spoof a stager check-in.

```
from itertools import product
import string
import sys
try:
    from crccheck.checksum import Checksum8
except ModuleNotFoundError:
    print("[~] crccheck not installed, please install it!")
    print("[+] pip install crccheck")

if len(sys.argv) != 3:
    print(f'[!] Usage: python {sys.argv[0]} <checksum_integer> <uri_length>')
    sys.exit(1)

charSet = string.ascii_letters + string.digits
rep = int(sys.argv[2])
while True:
    for wordchars in product(charSet, repeat=rep):
        uri = ''.join(wordchars)
        checksum = Checksum8.calc(bytes(uri, encoding='utf8'))
        if checksum == int(sys.argv[1]) and len(uri) == int(sys.argv[2]):
            print(f'[+] Calculated URI: {uri}')
            sys.exit(0)
    rep += 1
if rep > int(sys.argv[2]):
    print(f"[~] Unable to generate a URI with checksum of {sys.argv[1]} and length of {sys.argv[2]}")
    sys.exit(2)
```

Figure 15 – Python script to generate a URI of a specified length and Checksum8 value

This approach does have its limitations. If the adversary is using a stageless payload in their deployment, or if they have disabled staging altogether, we would not be able to retrieve a payload. A stageless payload is one that contains both the payload stage and its configuration in a self-contained package, so there is no requirement for a check-in to the C2 to complete the infection process (for more details see *Stager* vs *Stageless*).

PILLAGING TEAM SERVERS

At this point, we have one or even several data sources that are providing us with targets for scanning. We also have a means of generating the correct stager URI required for us to try and download a Beacon from each of the suspected Team Servers. Combining these two pieces of information, we can begin to automate the process of emulating a stager check-in to each of the potential targets and save any returned payloads to disk for subsequent processing and analysis.

```

def pull_beacon(baseurl, stage_url, headers, download_dir=os.getcwd()):
    # format beacon stage url
    beacon_url = f'{baseurl}/{stage_url}'
    try:
        # Try to pull down a beacon
        response = requests.get(beacon_url, headers=headers, timeout=10, verify=False)
        if response.status_code == 200:
            # Status Code 200 indicates we might have gotten a beacon
            print(f'[+] Potential Cobalt Strike server identified: {beacon_url}')
            payload = response.content

            # Calculate SHA256 hash of payload
            sha256 = hashlib.sha256(payload).hexdigest()
            print(f'[+] Payload/Beacon hash: {sha256}')

            # Extract IP from url
            ipaddress = beacon_url.split('/')[2].split(':')[0]

            # Extract Port from url
            port = beacon_url.split('/')[2].split(':')[1]

            # Generate filename based on Teamserver IP, Staging Port and Payload SHA256
            filename = f'{ipaddress}_{port}_{sha256}.bin'

            # Write payload to disk
            beacon_path = os.path.join(download_dir, filename)
            print(f"[+] Saving payload to disk: {beacon_path}")
            with open(beacon_path, "wb") as out:
                out.write(payload)

            return True

        print(f'[-] Server active but it did not return a payload: {beacon_url} ')
        return False

    except Exception as e:
        print(f'[!] Connection Error, server may be offline or port is inactive: {beacon_url} ')
        return False

```

Figure 16 - Python function to attempt to download a Cobalt Strike Beacon

PROCESSING

At this point, we should have some payloads ready for processing, although not all of them will be Cobalt Strike Beacons. We may have inadvertently hit on legitimate content or have been served spoofed content to mask the presence of an actual Team Server.

A valid Beacon payload stage is normally a Portable Executable (PE) file, most commonly a dynamic-link library (DLL), or position independent shellcode that will decode a PE. We can filter out false positives by removing files that are not PE files or that are smaller data files (less than 200KB) as Cobalt Strike payloads are generally larger than this size. What remains should be Cobalt Strike Beacons ready for configuration extraction.

BEACON CONFIG EXTRACTION

As already mentioned, the payload served by a Team Server takes one of two possible forms:

- 1. PE DLL (32 or 64 bit)
- 2. Shellcode (32 or 64 bit) that subsequently decodes and reflectivelyloads a pE DLL file in-memory

If we have been served the latter, then we will need to decode the encoded PE before we can attempt to extract the configuration data. This can be achieved in several ways, but two of the most common and easiest to scale are:

- Statically, by locating the encrypted PE payload and then decrypting it
- Dynamically, by emulating the shellcode so it self-decrypts, and then writing the decrypted PE payload to disk

Using either option requires an understanding of how the shellcode decodes the payload within.

The first block of the served payload is the decoder routine, which is responsible for XOR decoding the embedded PE. Immediately following the shellcode is a 32-bit XOR key that is used to decode both the payload size and the payload. Immediately after the XOR key is a 32-bit XOR encoded payload size, which is in turn followed by the variable-length, XOR encoded payload.

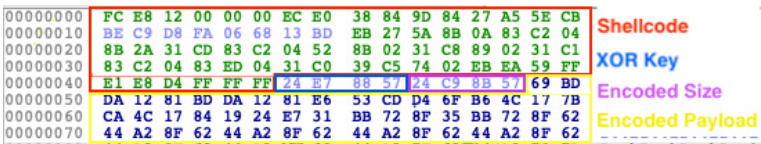


Figure 17 - Breakdown of a shellcode stager

When observed in IDA Pro (a popular disassembler), we can see a call to the decoder function immediately prior to the XOR key, with the encoded size and payload immediately following.

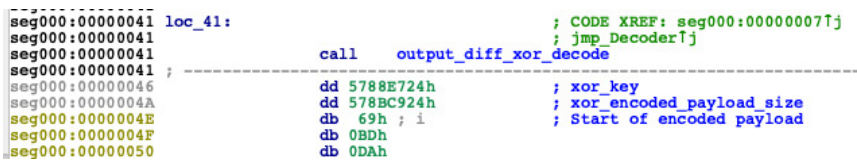


Figure 18 – IDA view of shellcode and other key components of the shellcode stager

This knowledge will prove useful later when we attempt to decode the payload.

Inspection of the decoder function reveals that it uses an output differential XOR routine to decode the Beacon payload. The XOR key is initially used to decode the first DWORD of the payload, and the updated to the value of the first decoded DWORD, which is then used to decode the second DWORD. This process is repeated until the entire payload is fully decoded, which is determined by the decoded size DWORD.

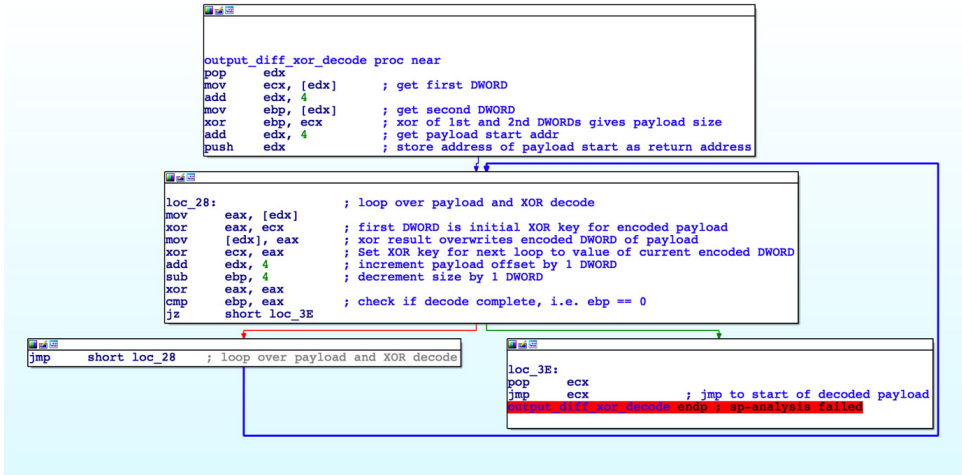


Figure 19 - Output Differential XOR decode routine

Now we are aware of the structure of the shellcode payload served-up by the Team Server and how it performs its decoding. We can develop some code to perform this statically, so that we can retrieve the Beacon payload for further processing.

```

def extract_beacon_from_shellcode(data):
    i = 0

    # 32bit Shellcode
    if data.startswith(b"\xfc\xe8"):
        # Locate the offset of the end of the decoder routine
        decoder_end = b"\xe8\xd4\xff\xff\xff"
        i = data.find(decoder_end)
        if i == -1 or i >= 0x50:
            decoder_end = b"\xe8\xd0\xff\xff\xff"
            i = data.find(decoder_end)

    # 64bit Shellcode
    elif data.startswith(b"\xfc\x48"):
        # Locate the offset of the end of the decoder routine
        decoder_end = b"\xe8\xc8\xff\xff\xff"
        i = data.find(decoder_end)
    if i == -1 or i >= 0x50 or i == 0:
        return False

    # Strip Decoder routine from encoded payload block
    data = data[i + 5:]

    # get payload length, XOR result of first 2 DWORDS after decoder
    data_len = int.from_bytes(data[:4], byteorder='little') ^ int.from_bytes(data[4:8], byteorder='little')

    # get encoded payload with second DWORD (XOR Encoded payload size) removed
    data = data[:4] + data[8:]

    # The initial XOR key is prepended to the data i.e. the first DWORD after the decoder block
    # XOR is Output Differential scheme i.e. XOR key is set to the value of the previous DWORD
    decrypted_beacon = bytearray([data[x] ^ data[x + 4] for x in range(0, data_len)])

    return decrypted_beacon

```

Figure 20 - Python function to decode a shellcode wrapped Beacon

This code first checks the shellcode's target architecture (32/64-bit) based on the opcode of the first instruction of the shellcode. The architecture will determine the pattern of the "call" instruction that should be located at the end of the decode function.

If a specified "call" pattern can be found, then we will have located the offset for the end of the decoder routine. Once we know this offset, we can then determine the values for the XOR key and payload size using their location relative to this offset. With the XOR key and payload size in hand, we can then perform the output differential XOR decode process, thereby retrieving the encoded Beacon payload.

The final stage in processing is the extraction of the config from the Beacon itself. Locating the config within a decoded Beacon can be performed quickly using known binary patterns commonly found in the config. Cobalt Strike Beacon configs have a particular structure and are single-byte XOR encoded, which causes the encoded config to have tell-tale patterns in its encoded form. The structure looks like this (using Kaitai Struct):

```
29     config_entry:
30         seq:
31             - id: index
32               type: u2
33               enum: index_names
34             - id: fieldtype
35               type: u2
36               if: index != index_names::done
37             - id: fieldlength
38               type: u2
39               if: index != index_names::done
40             - id: fieldvalue
41               size: fieldlength
```

Figure 21 - Cobalt Strike Config Entry Structure
(<https://gist.github.com/sixdub/a5361168ba7acecf7a7a214bf7e5d3d3>)

Index is a WORD (2-byte) value starting at offset 0x0 and is effectively the Setting ID number. The *fieldtype* is also a WORD, with three possible values:

- 0x1 for short
- 0x2 for integer (2 or 4 bytes)
- 0x3 for data or string

The value of *fieldtype* informs us of how to parse the *fieldvalue*. The *fieldlength* is also a 2-byte value indicating the length of the following data in the *fieldvalue*.

XOR encoding such a rigid collection of structures is highly susceptible to cryptanalysis, causing patterns to emerge in the encoded data. This is especially clear when those structures are XOR encoded using a non-null preserving XOR encoding scheme.

Depending on the version of Cobalt Strike Team Server that generated and served the Beacon in question, this will determine the pattern we need to search for. The default XOR key value is `0x69` for versions prior to version 4, and `0x2e` from version 4 onwards.

It is rare to see Beacons that have strayed from these default XOR values, but it can and does happen. In such cases, it would be necessary to brute force the XOR key value.

The first record in a Beacon's config is the 'Beacon Type' setting. The 'Beacon Type' entry in the config has an index of `0x1`, *fieldtype* of `0x1`, *fieldlength* of `0x2` and the *fieldvalue* varies depending on the Beacon Type.

This means that the first six bytes of the Beacon config remain static and have a hex value of `\x00\x01\x00\x01\x00\x02`. When this is encoded using XOR with a key of `0x69` or `0x2e`, we end up with two possible output patterns to search for to locate the start of a config block. These are `\x2e\x2f\x2e\x2f\x2e\x2c` and `\x69\x68\x69\x68\x69\x6b`.

If we find such a pattern, we will simultaneously know where the config starts as well as the value of the XOR key used in its encoding. Incidentally, this is also a useful "quick 'n dirty" means of distinguishing between versions 3.x and 4.x of Cobalt Strike.

If neither pattern is discovered, then we might be looking at one of the rare Beacons that are using a non-default XOR key. But we can still apply the same methodology in order to locate the encoded config by rotating through all possible single-byte XOR key values until a matching pattern is found.

Now that we can find the config within a decoded Beacon and have awareness of the XOR key as well as an understanding of the structure of each setting in the config, we can automate the final step of extracting the configuration. Once the configuration is parsed and dumped, we can store the results in a database for further analysis. SQLite is more than adequate for this purpose, but an ELK (Elasticsearch, Logstash, and Kibana) stack works well too. It all depends on what best serves your needs and fits with existing services and solutions.

Several Beacon config parsers are available and in common use by the security community. Here's a helpful and informative blog post about building a parser using Kaitai Struct.

Justin Warner

Using Kaitai Struct to Parse Cobalt Strike Beacon Configs

<https://sixdub.medium.com/using-kaitai-to-parse-cobalt-strike-beacon-configs-f5f0552d5a6e>

AUTOMATING THE HUNT FOR COBALT STRIKE

Now that we have defined our data sources, pillaged some Team Servers for shellcode and Beacons, and extracted their configs, we need to consolidate everything into an automated workflow. This automation can serve the needs of key stakeholders across all XDR products and services.

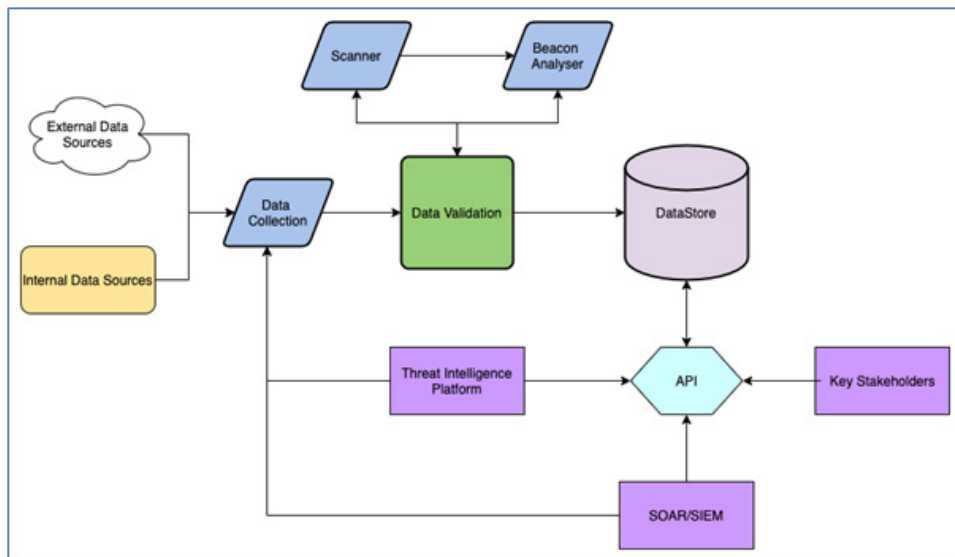


Figure 22 - Cobalt Strike hunting automation

Data collected from multiple sources will be passed through a validation process to provide a confidence rating based on several criteria, such as:

1. Source of data i.e., threat intelligence feed, Shodan, IR etc.
2. Confidence of search query (if any)
3. Overlap in detections i.e., same data from multiple sources
4. Beacon retrieved and processed

Once evaluated and scored, the data can be stored, disseminated, and acted upon as required. If necessary, this data could also be made accessible using an API so that it can be easily leveraged in future automation endeavors. When performed at scale, and over a long period of time, the resulting dataset can offer a wealth of information that can be used to bolster defenses, prevent breaches, and enhance intelligence.

But before we delve into the dataset closely, it's worth spending a moment familiarizing ourselves with Beacon's configuration and "malleable C2 profiles", so we can take stock of some of the data we have to play with.

