vb 2023 LONDON

4 - 6 October, 2023 / London, United Kingdom

C2F2: A FRAMEWORK FOR DETECTING C2 FRAMEWORKS AT SCALE

Sebastiano Mariani, Oleg Boyarchuk, Stefano Ortolani & Giovanni Vigna

VMware, UK & USA

smariani@vmware.com
boyarchuko@vmware.com
ortolanis@vmware.com
vigna@vmware.com

ABSTRACT

Command-and-control (C2) frameworks are systems used to remotely manage and maintain access to compromised devices. C2 frameworks are routinely used for legitimate penetration testing, to evaluate the security of a system and its ability to detect attacks, with the goal of helping organizations improve their defences and better protect against real-world attacks.

To simulate the polymorphic nature of modern threats, C2 frameworks often support the creation of custom payloads that can be used to simulate specific types of attacks. These payloads can be used to test the effectiveness of endpoint security products, network-based intrusion detection systems, and other types of security measures.

Unsurprisingly, cybercriminals immediately took advantage of the opportunity to use effective and well-maintained penetration testing tools for malicious purposes and, as a result, frameworks such as Cobalt Strike, Metasploit and Sliver have increasingly been used in the process of breaking into enterprise networks.

This report analyses a number of C2 frameworks that are used by cybercriminals, focusing the analysis on the implants that are deployed as bridgeheads in compromised networks.

In particular, *VMware* TAU developed a 'framework for the analysis of C2 frameworks', called C2F2, that leverages the ability of C2 frameworks to generate custom implants to automatically synthesize a large dataset, which is then used to generate detection procedures based on machine-learning techniques.

As part of this research, we release C2F2 on *GitHub*, and provide enough plug-ins to instrument all C2 frameworks detailed in this report. Each plug-in is also paired with instructions on how to programmatically explore large spaces of possible configuration options when generating the implants. To conclude, we also provide the dataset of all generated implants for each analysed C2 framework.

INTRODUCTION

Cybersecurity breaches are becoming increasingly sophisticated, with cybercriminals employing multi-stage attacks that involve initial penetration, establishment, and lateral movement to expand their control over the target's assets and resources.

While much attention has traditionally been focused on detecting and preventing the initial breach, it is essential to recognize that post-breach behaviour provides significant opportunities for detection and containment. Command-and-control (C2) frameworks, which are originally created with the legitimate purpose of testing the security solutions deployed in an organization's network, are increasingly being leveraged by cybercriminals for post-breach activities due to their effectiveness and well-maintained nature. These frameworks enable cybercriminals to gain persistent control over compromised systems and execute various malicious activities, such as exfiltrating data, moving laterally, launching further attacks, or maintaining access for future use.

Efforts have been made to develop detection mechanisms for popular C2 frameworks, focusing primarily on communication with the controlling server. These detection techniques typically rely on known patterns of communication or specific signatures associated with well-known C2 frameworks. However, these efforts have prompted cybercriminals to adopt less well-known frameworks, such as Sliver, and also evade detection by leveraging different communication techniques or obfuscation methods offered by several C2 frameworks, such as the 'malleable C2' communication mechanism offered by Cobalt Strike. As a result, there is a growing need for improved detection of C2 frameworks, including the less well-known ones that may be used by cybercriminals to avoid detection.

An opportunity for detection of post-breach activity is the identification of C2 implants. These are the components that are deployed on a compromised host to maintain access to the target network and operate as a bridgehead for further attacks. Unfortunately, most C2 frameworks offer polymorphic mechanisms to customize and obfuscate the implants, making their identification and classification challenging. However, VMware's insight is that the very same mechanisms that are used to generate custom implants can be leveraged to generate thousands of implants by fuzzing the parameters of the implant generation process. Using this process, it is possible to generate large-scale datasets that can then be fed to a machine-learning-based analysis pipeline to identify features and characteristics supporting the detection and classification of C2 implants that operate within the network perimeter.

This report aims to provide a comprehensive survey of several of the most commonly used C2 frameworks, both commercial and open-source, that are currently being utilized by cybercriminals. The report will also introduce a novel approach for the collection of a large dataset of implants, which can support effective detection of these artifacts using machine-learning approaches. Additionally, the report describes a detection and classification approach based on machine learning, with the goal of contributing to the advancement of C2 framework detection and containment strategies in the field of cybersecurity, providing valuable insights for practitioners and researchers alike.

A C2 FRAMEWORK MENAGERIE

When it comes to C2 frameworks, the number of active projects (more than 120 at the time of writing) is so overwhelming that in 2020 SANS Institute began maintaining a list to aggregate all frameworks publicly available (either open-source or

commercial), called C2 Matrix [1]. To further assist security control testing and training exercises, the C2 Matrix project has released the SANS Slingshot C2 Matrix VM, a bespoke virtual machine containing a set of pre-installed C2 frameworks in an attempt to further lower the cost of entry for red, blue, and purple teams alike [2].

Since our analysis is focused on highlighting detection opportunities by focusing on the C2 implants, this report is limited to C2 frameworks that are able to provide some degree of customization when generating the implants (either in terms of functionalities or embedded features). Out of the resulting set, we selected 10 C2 frameworks representative of either what has been used in the wild by malicious threat actors, or what has been included in distributions such as Kali or the SANS Slingshot C2 Matrix VM. As for commercial C2 frameworks, our analysis is limited to those frameworks actively used by threat actors and that have been leaked on *VirusTotal* or *GitHub* ([3] and [4] for Brute Ratel, [5] and [6] for Cobalt Strike). Table 1 provides the full list.

Name	Licence	Website	Slingshot	Kali	In the wild
Cobalt Strike	Commercial	https://www.cobaltstrike.com/			Y
Metasploit	BSD3	https://metasploit.com	Y		Y
Sliver	GNU GPL3	https://github.com/BishopFox/sliver		Y	Y
Brute Ratel	Commercial	https://bruteratel.com/			Y
Covenant	GNU GPL3	https://github.com/cobbr/Covenant	Y	Y	Y
Merlin	GNU GPL3	https://github.com/Ne0nd0g/merlin	Y	Y	Y
Shad0w	MIT	https://github.com/bats3c/shad0w			Y
Empire	BSD3	https://github.com/BC-SECURITY/Empire	Y	Y	Y
PoshC2	BSD3	https://github.com/nettitude/PoshC2	Y	Y	Y
Godoh	GNU GPL3	https://github.com/sensepost/godoh		Y	

Table 1: List of selected C2 frameworks.

Cobalt Strike

Cobalt Strike [7] is a commercial, full-featured penetration testing software that provides an integrated platform for adversary simulations and red team operations. It includes a command-and-control (C2) framework that allows an operator to interact with compromised systems and perform various tasks, such as lateral movement, data exfiltration and privilege escalation.

The Cobalt Strike C2 framework includes several key components:

- The Beacon payload: This is a small, lightweight piece of malware that is used to establish a connection between the compromised system and the Cobalt Strike C2 server.
- The C2 server: This is the central component of the Cobalt Strike C2 framework, which receives commands from the operator and relays them to the Beacon payloads.
- The Cobalt Strike client: This is the software that the operator uses to interact with the C2 server and manage the compromised systems. The client provides a graphical user interface (GUI) for the operator to perform various tasks and view the results. A screenshot of the GUI is presented in Figure 1.

One of Cobalt Strike's most powerful features is the 'Malleable C2' feature, which allows users to customize the network traffic generated by the Cobalt Strike Beacon payload, to make it appear like legitimate traffic or to bypass specific security controls. It achieves this by allowing users to define custom profiles that specify how the Beacon should communicate with its C2 server.

Malleable C2 profiles are written in a domain-specific language that allows users to customize various elements of the Beacon's network traffic, such as user-agent strings, HTTP headers, and other parameters. By modifying these elements, users can make the Beacon's traffic blend in with legitimate traffic, making it harder for security tools to detect it.

For example, a user could create a Malleable C2 profile that modifies the user-agent string to match a legitimate browser or that modifies the HTTP headers to mimic traffic from a specific website that is often used in the network being targeted.

In our framework, the process of generating a beacon involves several steps. First, a third-party tool known as C2concealer [8] is used to create a Malleable C2 profile. Second, a Cobalt Strike server is started using the generated C2 profile. This server will act as a command-and-control centre for the implant once it is deployed on the target system. To generate the beacon binary itself, two scripts written in the Aggressor Script [9] language are used. The first script creates a listener on the server for the beacon to connect to, and the second script generates the binary itself. The resulting binary will use the settings specified in the Malleable C2 profile as its communication protocol, making it more effective in evading detection.

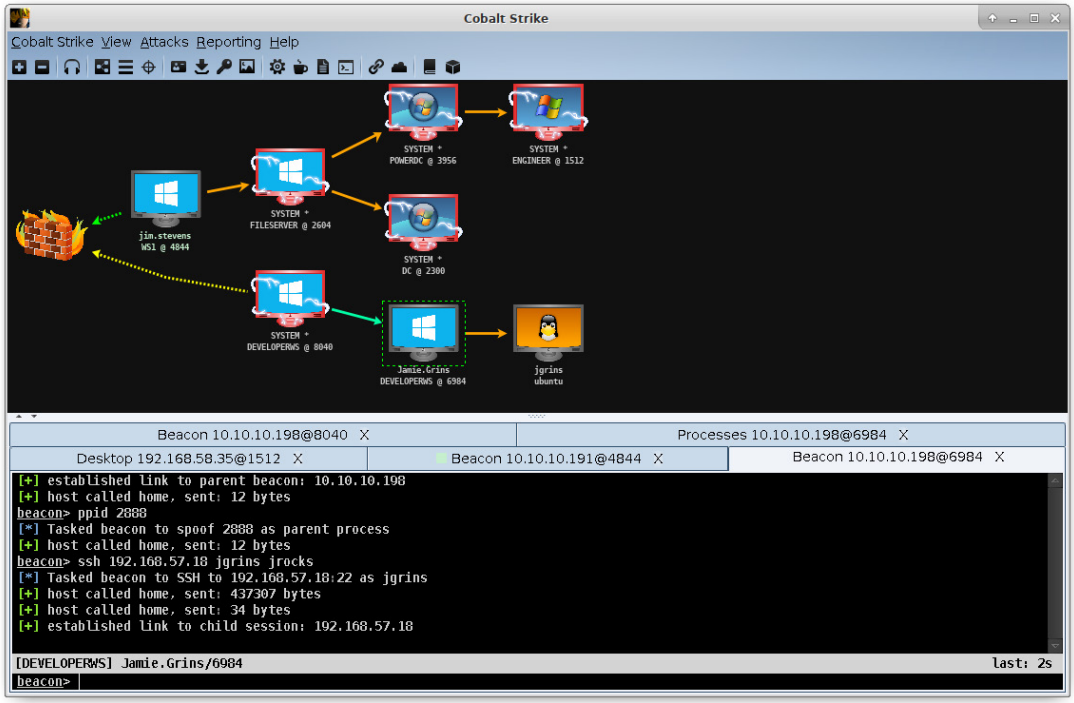

Figure 1: Cobalt Strike console screenshot (source: [7]).

Cobalt Strike is the most sophisticated and popular C2 framework, and therefore it is not surprising that pirated versions are routinely used by threat actors while compromising target networks (see for example [10] and [11]).

The use of Cobalt Strike for malicious purposes has become so widespread that, in November 2022, *Google* published a number of Yara rules to detect pirated versions of Cobalt Strike [12] to help stop the abuse of this security tool.

Metasploit

The Metasploit Framework [13] is an open-source penetration testing tool that provides tools for compromising a target, installing an implant, and remotely controlling the implant to perform further exploitation and lateral movement.

The Metasploit Framework allows for the creation of 'stagers', which are small programs whose goal is to deliver a more complex payload. The most commonly used payload is Meterpreter.

Meterpreter is typically used after a successful remote code execution exploit to gain persistent control over a target machine. It provides a powerful and flexible interface for interacting with compromised systems, and can operate on *Windows*, *Linux* and *MacOS*. It is designed to be stealthy and hard to detect, making it an ideal tool for post-exploitation activities such as privilege escalation, lateral movement and data exfiltration.

Once Meterpreter is installed on a target system, an attacker can use it to perform a wide range of tasks, such as viewing and manipulating files and processes, capturing keystrokes and screenshots, and even launching other Metasploit payloads. Additionally, Meterpreter provides a powerful scripting environment that allows attackers to automate tasks and perform complex actions on compromised systems.

Metasploit includes a standalone payload generation tool, *msfvenom*, that allows one to create customizable implants that are based on Meterpreter payloads, e.g. `windows/meterpreter/reverse_tcp` (back-connect to the attacker over TCP), `windows/meterpreter/reverse_http` (back-connect to the attacker over HTTP), `windows/meterpreter/reverse_https` (back-connect to the attacker over HTTPS). The payload-specific configuration data (e.g. the C2 address and port), the output file format (e.g. EXE, DLL, MSI), the target CPU architecture (e.g. x86, x64), and many other parameters are passed as arguments to msfvenom. With the help of the `--list` parameter in combination with `payloads`, `formats` or `archs`, it is possible to retrieve the full list of available Meterpreter payloads, output file formats and the target CPU architectures, respectively.

Sliver

Sliver [14] is an open-source, cross-platform C2 framework (it supports *Windows*, *Linux* and *MacOS*) written in Go. Sliver is designed to provide a flexible and modular platform for conducting post-exploitation activities on compromised systems. It is often used by penetration testers, red teamers, and security researchers.

Sliver allows for the generation of 'implants' that support the remote management of compromised hosts. The implants connect back to the C2 server to receive commands and perform additional attacks. Sliver supports a number of mechanisms for the communication between an implant and a C2 server, including DNS, Mutual TLS (mTLS), WireGuard, and HTTP(S). In addition, the implants can operate in either 'beacon mode' or 'session mode'. In the former case, the implant connects back to the server at specific intervals to retrieve tasks to perform, while in the latter case the implant maintains an interactive session with the server.

Figure 2 shows the command-line interface provided by the Sliver server.

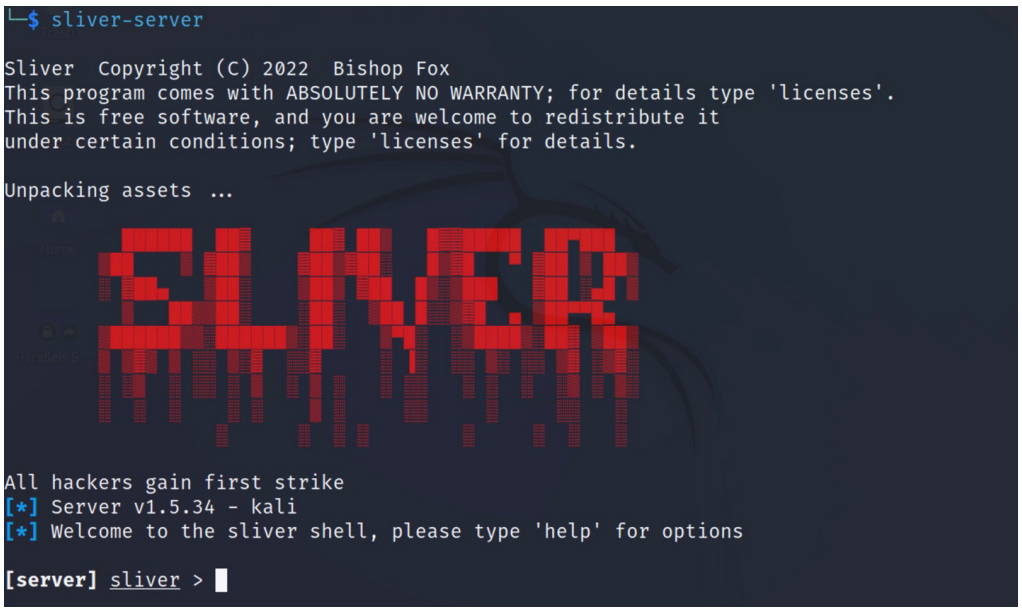

Figure 2: Sliver server interface.

The generate command in the Sliver CLI generates customizable implants. Parameters, such as the communication protocol (HTTP(S), mTLS, WireGuard or SMB), the target OS (*Windows*, *Linux* or *Mac*), the target CPU architecture (x86 or AMD64), the output file format (standard or service EXE, DLL or shellcode), and many others can be passed as arguments. One of the interesting features of Sliver is the support of DNS canaries, which are unique per-binary domains that are optionally inserted during the string obfuscation process. These domains are deliberately not obfuscated so if they are ever resolved, one will receive an alert about which specific file was discovered by the adversary blue team. Another feature that makes Sliver implants stand out is the ability to include specific limitations. For example, it is possible to configure an implant so that it will refuse to work after a specific date, or to limit its actions to a system with a specific locale, username, or host name.

Sliver has routinely been used by threat actors in various campaigns (see for example, the February 2023 attack [15] and also [16, 17, 18]).

Brute Ratel

Brute Ratel [19] is a commercial command-and-control (C2) framework written in C# that is designed to provide a flexible and modular platform for conducting post-exploitation activities on compromised systems. At the time of writing this paper Brute Ratel does not provide exploitation modules (like Metasploit does) or vulnerability scanning capabilities.

The Brute Ratel implant is called a 'badger'. Implants can communicate with the C2 server and with other badgers using HTTP, HTTPS, DNS over HTTPS, and SMB. The implants operate asynchronously, asking the server for tasks, performing them, and returning the results. Badgers provide support for a number of activities, from gathering information (e.g. by running Mimikatz) to social engineering attacks that collect credentials on the local system.

Figure 3 shows the Brute Ratel console.

The communication between the two components of the Brute Ratel framework can be customized using communication profiles, which allow for the modification of basic aspects of the communication.

The author of Brute Ratel has introduced many evasion techniques to prevent detection, such as the use of indirect system calls, the ability to unhook the callbacks set by EDR tools, as well as stack and heap encryption.

Badgers can be generated via the client UI as well as by communicating with the server directly, using Brute Ratel's proprietary protocol. The payload profiles contain such parameters as the protocol (TCP, HTTP, SMB or DoH), the set of C2 addresses and ports, the protocol-specific configuration (e.g, the name servers in the case of the DoH protocol; the SMB

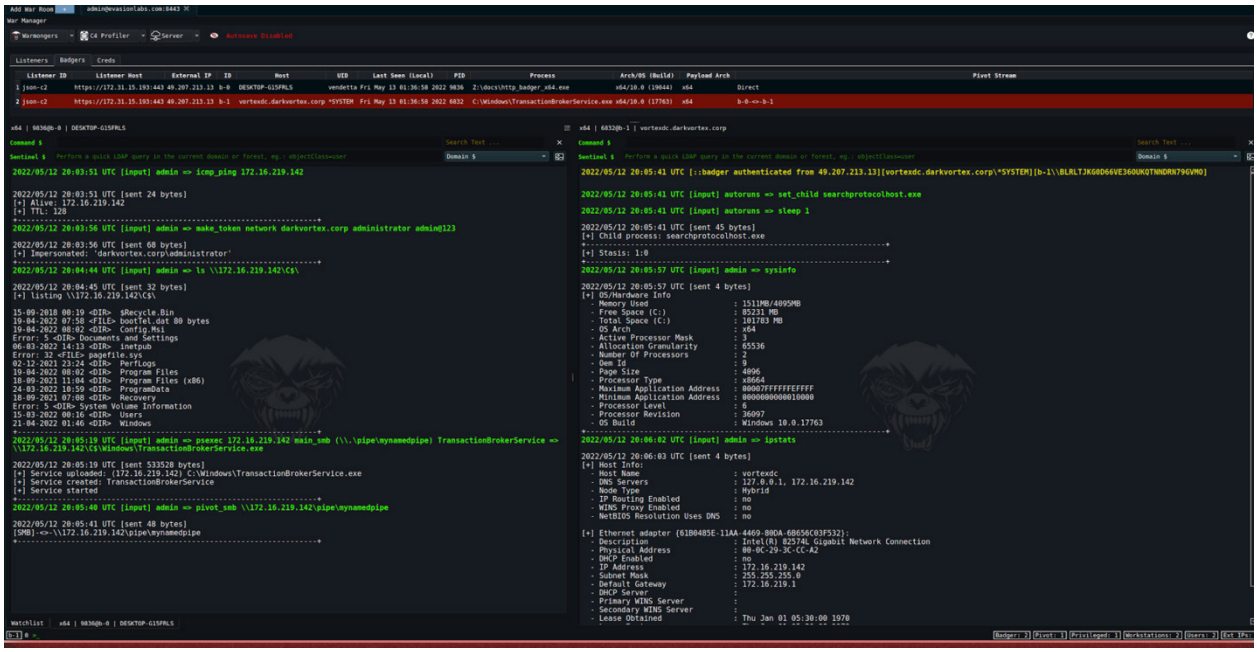

Figure 3: Brute Ratel console.

pipe name in the case of the SMB protocol; custom User-Agent and extra headers for HTTP requests to blend in with the benign traffic), the sleep interval between the communication attempts, the proxy address, and many more. The resulting file can target different CPU architectures (x86 or x64) and they can be delivered in different formats (shellcode, DLL or EXE, and built as a service executable).

While Brute Ratel has been designed and developed to be used in adversarial simulations and penetration tests, the framework has been used by threat actors in carrying out attack campaigns, such as ransomware attacks [20] and others [21].

Godoh

Godoh [22], as the name suggests, is a C2 framework written in Golang characterized by using DNS-over-HTTPS as a communication protocol. DNS-over-HTTPS (DoH) is a recent standard [23] for sending DNS queries (normally exchanged in clear text) over an encrypted TLS channel, using HTTP as a protocol to map DNS requests and responses. This is an extremely attractive feature from an attacker's perspective, as all C2 communication will look like normal (and legitimate) DNS-over-HTTPS traffic.

Implementing a C2 protocol over DoH requires tunnelling typical C2 requests and responses (usually containing the output of commands, or a request for further instructions) into DNS queries which are designed to resolve domain names to IP addresses. While the request can easily be mapped to a specially crafted domain to be resolved (and, to be more specific, encoded into a sufficiently long sub-domain whose domain is under the control of the attacker), the response typically requires more data than a regular IPv4 or IPv6 address. While the response to a query to resolve A records (and AAAA records for IPv6) is clearly limited in size (as dictated by the number of bytes required to store the resolved IP address), the DNS protocol allows a special type of record, called TXT, to hold up to 255 arbitrary characters, making it the perfect candidate to encode a (portion of) an arbitrary response.

When DoH is combined with DNS tunnelling, the attacker would just need to control a name server responsible to answer queries for a specific (attacker-controlled) domain. The overall network traffic would look like normal HTTPS traffic to a legitimate DoH provider. Godoh implements several modules to use different providers – *Google*, *Cloudflare* and *Quad9* to name a few. Note that the overall traffic would still disclose the hostname of the provider chosen to serve DoH requests (for example dns.google.com), making the overall traffic stand out were the implant to be deployed in an enterprise setting configured with a different DNS provider. To prevent this, Godoh also includes an additional DNS client that uses domain fronting via google.com to relay queries to dns.google.com, making the overall DoH traffic to dns.google.com indistinguishable from HTTPS traffic to google.com. As of April 2019, however, *Google* has disabled this functionality.

From an implant perspective, Godoh is quite simple, and it does not have the plethora of modules and commands that more advanced frameworks have. The implant is able to send and receive files, and it can be compiled against several DNS client modules to support as many DNS providers as the client. When generating an implant, it is only possible to change the

address or domain used by the C2 server. In our analysis, we generated implants for all major DNS providers, and also configured the 'address' option with random, but legal, values.

The TLS connection (used by the HTTP protocol) is implemented via the standard TLS library of Golang, making the resulting JA3 hashes too common to be used as a technique to recognize HTTPS connections established by this C2 framework. Godoh is developed by *SensePost*, a team of researchers employed by *Orange*, and it was released for the first time in 2018.

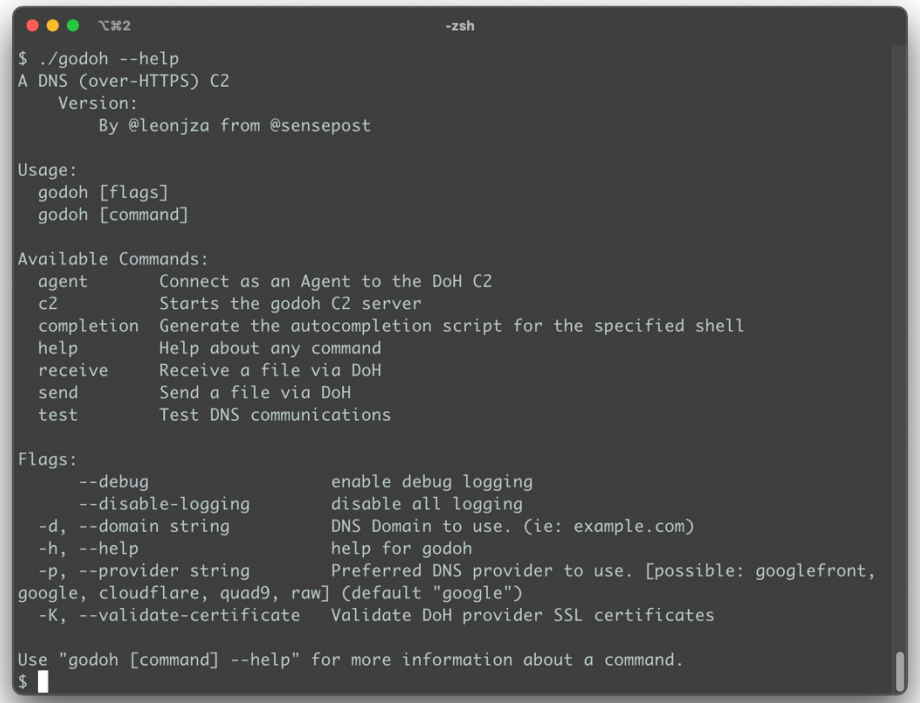

Figure 4: Available commands from a freshly compiled binary.

Shad0w

Shad0w [24] is a modular C2 framework (the C2 panel is shown in Figure 5), published for the first time in April 2020 by *JumpSec LABS*. It is written in Python 3 and C and relies on an external payload generation framework called Donut to create the implants (like Cobalt Strike, shad0w adopted the term 'beacon' when referring to implants). The whole framework is distributed with the Dockerfiles required to compile and execute the framework regardless of the underlying operating system (cross-compilation on architectures other than x86 is still not supported, however, including relying on Docker's ability to emulate the x86 architecture via the switch '`--platform linux/amd64`' when running on ARM).

By using Donut, implants can execute arbitrary .NET assemblies, executables, JavaScript files, VisualBasicScript files, or even XSL files, directly from memory. The resulting payload can be generated in different formats: a normal PE file, a PowerShell script, or even just shellcode. The implants provide several primitives to perform process injections and are designed to evade EDR and anti-virus solutions by relying on several techniques, such as directly using syscalls rather than invoking higher-level API calls, which are normally targeted by user-land hooks. Another important feature inherited by Donut is the ability to encrypt the entire payload, making automated static analysis more challenging.

Communication back to the C2 server is done via HTTPS; it is also possible to specify custom user agents or introduce arbitrary jitter. While custom user agents are already protected by the encryption provided by HTTPS (unless TLS is decapsulated), the ability to introduce arbitrary jitter is a valid approach to foil traffic analysis as often employed by NDR solutions. It is also possible to use the 'shrink' option to further reduce the size of the implant. In our analysis, we randomized all these options, and created a different agent for each legal combination. An interesting feature implemented by the C2 server is the ability to create live mirrors of real websites (for example CNN), such that C2 traffic is generated only when contacted by the beacon, while normal web browsing ends up generating traffic that appears to be legitimate.

The framework itself is highly modular and allows for the easy deployment and execution of standalone programs, such as Mimikatz, SharpSocks, or other well-known offensive tools. Another feature worth mentioning (and testament to the framework's claims of modularity) is the ability to elevate privileges directly from the C2 console by relying on built-in exploits.

Figure 5: Shad0w's C2 panel.

While development stopped in August 2021, the framework is still included in the 2022 SANS Slingshot VM [25] and actively taught in training and security courses. We are not aware of any high-profile threat actor employing shad0w, or even incidents caused by any of its implants. The underlying techniques do, however, underpin many of the tools routinely employed in high-profile ransomware attacks [26].

Empire 3

Empire [27], like Cobalt Strike, is one of the most well known post-exploitation frameworks available. Unlike the framework sold by *Fortra*, however, it is freely available and open-source, making it the framework of choice for both well-funded and opportunistic threat actors alike. Maintained by *BC Security* [28], it recently reached its fifth major version, featuring a renovated REST API, an official web application as a GUI (Starkiller), and a code base fully compatible with Python 3.

Empire's architecture is extremely modular and even the C2 server is structured into two client/server components, thereby supporting external clients and/or user interfaces (see Figure 6 for the default client). The C2 server supports more than 400 tools or commands written in PowerShell, C#, and Python that can be executed once the agent is successfully deployed. The agent, in turn, is again written in either PowerShell, C#, or Python, and connects back to the C2 server either via an encrypted HTTPS channel (configurable via malleable C2 profiles) or to any other deployed listener (Empire includes listeners to proxy C2 communications via third-party services like *OneDrive* or *Dropbox*). The C2 server can be further configured to randomize JA3/S and JARM fingerprints, considerably raising the bar for threat hunters attempting to track the attacker's network infrastructure using services such as *Censys* or *Shodan*.

Empire's agent works in synergy with another component called a stager. Stagers are bespoke loaders that are ultimately responsible for loading the agent. Stagers can be macros, PowerShell one-liners, HTA files, or just shellcode. Choosing the right stager depends on the target operating system and application, and Empire tries to accommodate this need by providing a wealthy collection ready to be used. Like shad0w, Empire can also rely on Donut [29] to generate evasive or encrypted shellcode.

When generating the stager (and hence the agent to be deployed), it is possible to configure the user agent, the proxy, whether to obfuscate the agent, the number of retries between connections, as well as the kill date of the implant and the connection delay. Remember that generating a stager also requires the configuration of the listener, which includes the host, the headers, and the staging key, but also whether to evade JA3 fingerprinting. In our analysis, we generated legal but random combinations for all these parameters.

Once the agent is deployed, a connection is established with the listener running on the C2 server. Each agent used to be able to connect only to one listener, and only with configuration, and keeping listeners and agents in sync was the task of the campaign operator. Since 2018, it has been possible to configure multiple listeners within the same agent, making a campaign using Empire substantially more resilient. It is still necessary, however, to embed the configuration of the listener inside the agent whenever the listener is updated (for example by using another malleable C2 profile).

Empire 3 is a mature C2 framework, and as such, it is often used by threat actors to propel their campaigns. For example, in September 2022, CISA produced an alert about the Vice Society threat actor launching a set of campaigns targeting the education sector, especially kindergarten and K12 institutions [30]. The *OneDrive* listener was also reportedly used by Obscure Serpens, in a campaign that targeted government agencies in Middle Eastern countries [31] and used spear-phishing to deliver malicious *Microsoft Office* documents.

While extensively adopted, threat actors are quick to change their toolset if need be. In a recent report detailing a set of ransomware campaigns targeting high-value targets, researchers [31] noted how the FIN12 group had slowly been migrating their operations to Cobalt Strike, possibly because of their higher rate of update, even though, unlike Empire 3, the C2 framework is not freely available.

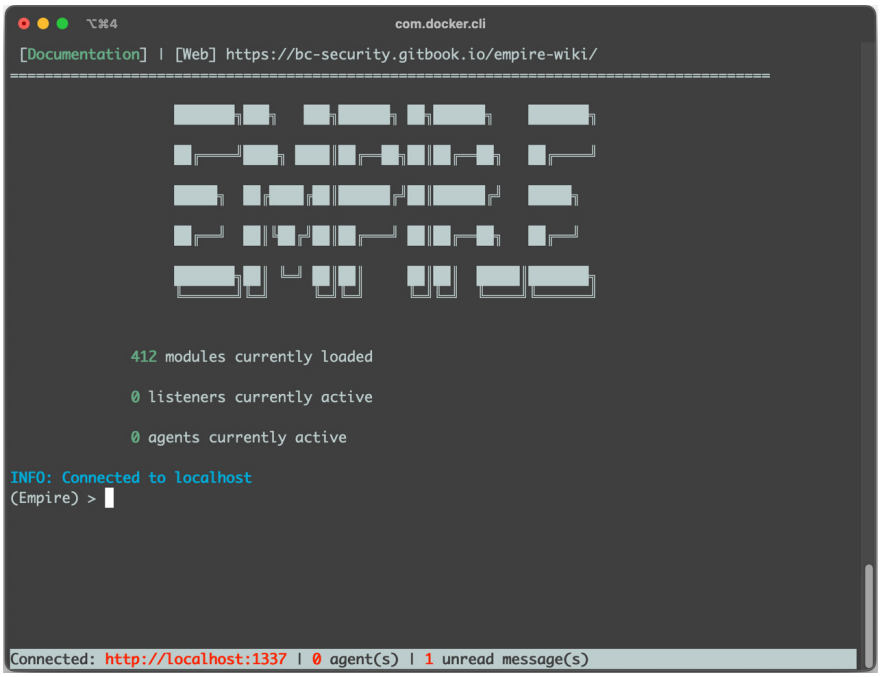

Figure 6: Empire 3 client after loading.

Merlin

Merlin is a post-exploitation C2 framework written in Golang [32]. One of its major selling points is its capability to run both server and agent on multiple architectures as long as Golang can run natively (making it a mandatory choice when targeting ARM systems). Merlin supports a hefty number of C2 protocols, ranging from HTTP/1.1 to HTTP/3. Development started in 2017, and it has been available on *GitHub* [32] ever since (the command line interface is shown in Figure 7).

Merlin is the C2 framework with the highest number of supported C2 protocols, including HTTP/2 and HTTP/2 over QUIC, making it the perfect choice to test the ability of IDS systems to parse even the newest protocols. The agent traffic is also encrypted using JSON Web Encryption [33], while mutual authentication is done via OPAQUE [34]. Merlin also tries to evade standard beaconing detection techniques by allowing each message to be randomly padded up to a given size. Whenever TLS is used, it is also possible to specify the desired JA3 fingerprint, to further evade NDR systems.

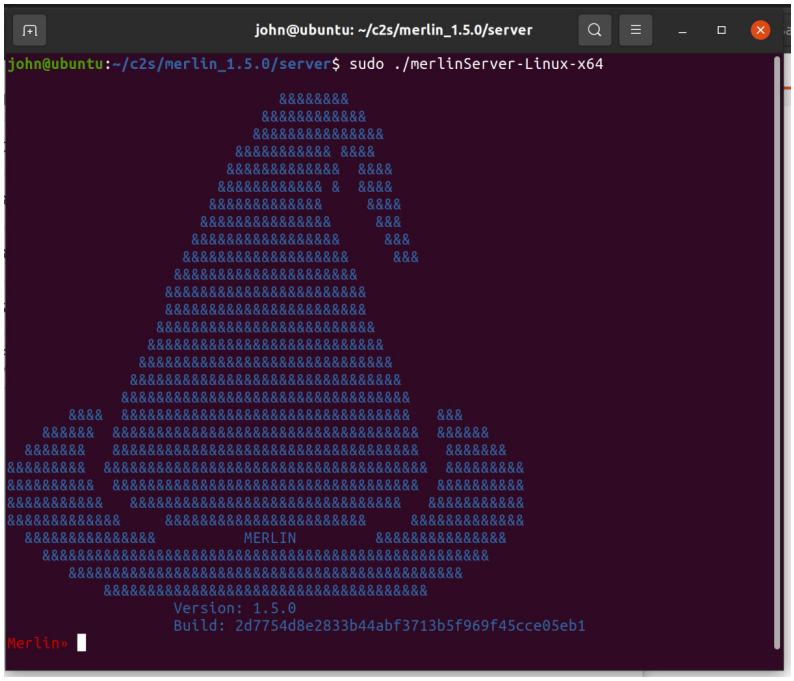

Figure 7: Merlin CLI.

The agent supports Donut (and other tools) to provide a degree of obfuscation and evasion; it also implements several techniques to run the shellcode (from creating a remote thread, to inserting craftier asynchronous procedure call objects to the queue of existing threads). Like more mature C2 frameworks, the pool of commands and scripts that can be run by the agent can be extended by creating additional Merlin modules; the *GitHub* repository comes with a collection of modules targeting the most common operating systems (albeit just on x64).

Some of the configuration options required by the agent are quite standard: the URL of the C2 (including the port), a kill date (when to stop operations), the maximum of retries when connecting, and how long to sleep between attempts; unlike other frameworks however, it is also possible to specify the JA3 hash and the proxy. In our analysis we randomized all these settings to generate our data set of x64 *Windows* agents.

The number of advanced network capabilities makes Merlin the C2 framework of choice for campaigns aiming to fly under the radar. This is exactly what researchers from *Bitdefender* discovered while investigating a malicious campaign (likely operated by a Chinese threat actor) targeting telecommunications companies in the Middle East [35] where the threat actor even compiled a version of the agent in which all the most characteristic strings had been removed.

PoshC2

PoshC2 [36] is a C2 framework written in Python 3 specialized in post-exploitation, lateral movement, and described as 'proxy-aware'. It is developed by *Nettitude*, and it has been active since 2018. The agents/implants are implemented in PowerShell, C#, and Python 3. The C2 servers support HTTP and HTTPS (encryption is still applied even when using HTTP), and it also provides auto-generated rewrite rules to shield the C2 API endpoints from unwanted traffic, thereby adding a degree of operational security to the C2 infrastructure. Figure 8 shows the output of the C2 server right after the first execution.

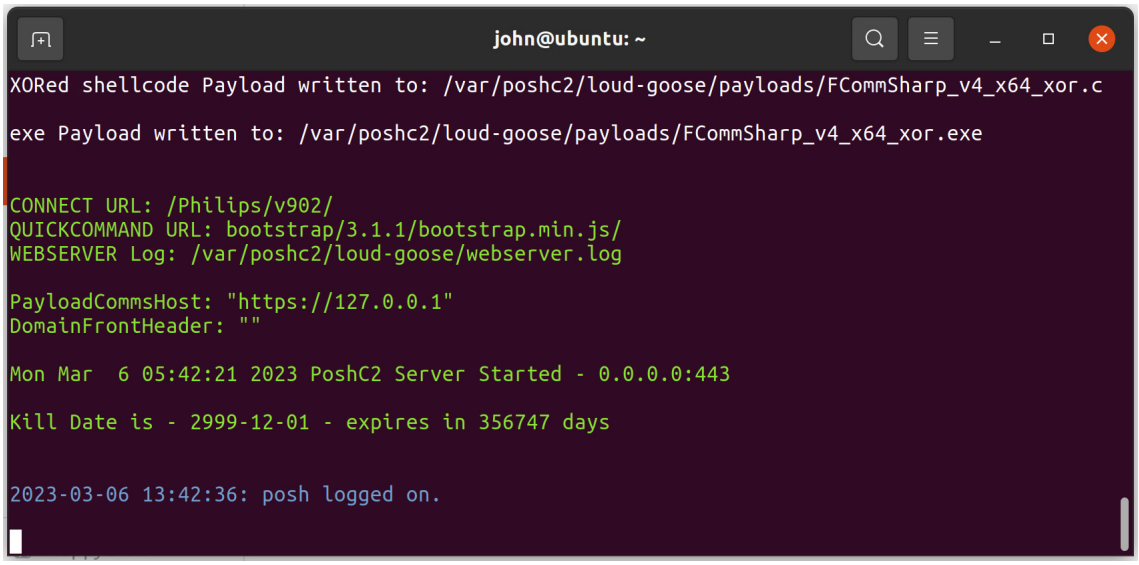

Figure 8: Output of PoshC2 server once launched.

Agents have unique capabilities, like the ability to daisy chain the network traffic through another implant, and can optionally contain hard-coded proxy credentials, showcasing the determination of PoshC2 developers in targeting inhospitable enterprise environments. While there are a number of different implementations, it should be noted that the implementation affects what functionalities are available: for example, only the C# implant can execute entire modules from memory (like GhostHound and BloodHound). Further, agents are limited by the available runtime: for example, an implant implemented in Python will require the presence of the Python interpreter on the targeted system.

At generation time, the agent requires some mandatory options to be configured. In our analysis we generated payloads with random, but legal, combinations for the URL and port of the C2 server, the sleep intervals between connection attempts, the jitter of the connection to the C2 server, and the termination date of the implant.

Note that PoshC2 already generates over 150 payload modifications given a single set of parameters to provide a degree of evasion.

Besides offering smart C2 proxying capabilities, PoshC2 is famous for deeply integrating with SharpSocks, an open-source SOCKS server, useful to expose to the attacker subnets that would not normally be directly accessible. This allows the attacker, for example, to establish RDP connections to internal hosts that are not publicly exposed.

PoshC2 is an extremely mature framework, and as such is widely abused by threat actors, like APT33 [37]. PoshC2 has recently been reported as the tool of choice for attacks against the financial industry in West and North Africa [38] . Unlike

other frameworks, and as documented in [39], PoshC2 does not require the changing of any defaults such as the TLS certificates, meaning that it is still possible to hunt for rogue C2 servers using *Shodan* or *Censys* [40].

COVENANT

Covenant [41] is an open-source command-and-control framework written in C#. Since it uses .NET Core [42], Covenant can operate on multiple platforms (*Windows*, *Linux* and *MacOS*).

The implant provided by the framework is called a Grunt, and specific 'listeners' can be used to specify how the implants communicate with the C2 server. Once a host is compromised, the Grunt connects back to the C2 server and awaits instructions.

The attacker can ask a Grunt to perform a 'task' (e.g. run Mimikatz on the compromised host to collect the credentials of local users). When a task is assigned, a corresponding binary package is created and sent to the Grunt for execution. By doing this, Covenant minimizes the amount of code that could be detected by a security solution. In addition, Grunts are obfuscated with ConfuserEx, to prevent detection by static signatures.

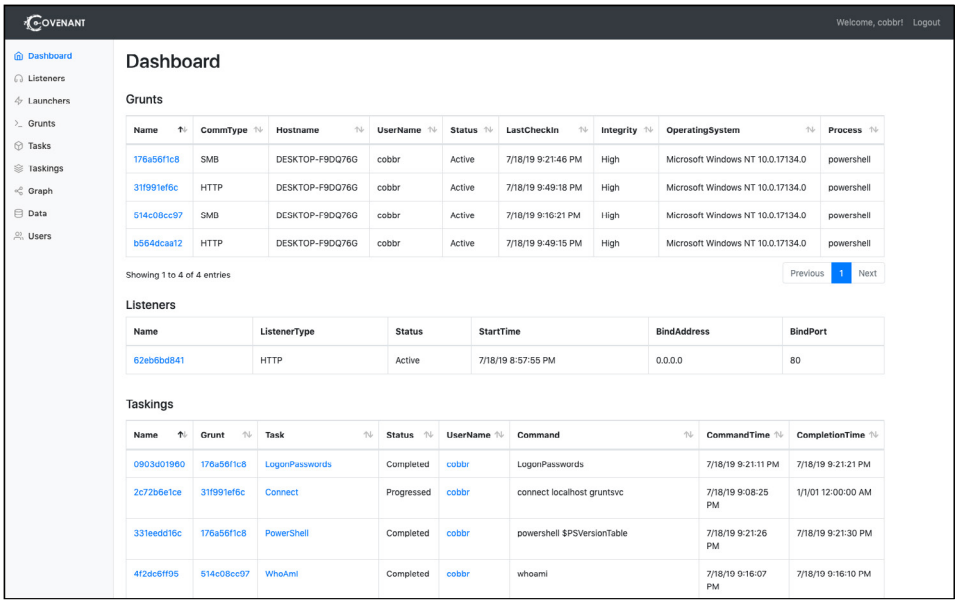

Figure 9: The Covenant dashboard.

The Covenant C2 framework also includes a web interface, which allows multiple attackers to interact with compromised systems using a user-friendly web interface. The web interface provides an overview of all active sessions and allows operators to view and modify session properties and request the active Grunts to execute tasks.

Grunts must be executed with the help of one of the launchers that Covenant supports. The most basic launcher is called the Binary launcher, which is a standard EXE file. Other launchers use more sophisticated techniques to run a Grunt, e.g. the MSBuild launcher uses msbuild.exe to launch a Grunt using an in-line task. When creating a launcher, one needs to pick an implant template and a listener that has been created earlier. There are four default implant templates: GruntHTTP (a *Windows* implant that communicates over HTTP), GruntSMB (a *Windows* implant that communicates over SMB), GruntBridge (a customizable implant that communicates with a custom C2Bridge), and Brute (a cross-platform implant built on .NET Core). There are only two listener types available: HTTP and Bridge. Specific details of a listener, such as the bind address, the port, and whether to use SSL or not, are among those that must be specified during listener creation.

Covenant has been used in the wild to control compromised hosts (see for example [43]).

IT'S RAINING IMPLANTS

Implants play a critical role in C2 frameworks by enabling threat actors to establish a foothold in a target environment and execute their malicious objectives. One of the significant challenges in detecting these implants is their polymorphic nature, which allows them to adapt and modify their code and communication protocols to evade detection by security systems. This can make it challenging to develop reliable and effective signatures for detecting these implants. However, in our C2F2 framework, we exploit the polymorphic capabilities of these implants to create a large dataset of samples. By applying a custom machine-learning pipeline to this dataset we were able to identify patterns in the behaviour of these implants, which can then be used to develop more effective detection strategies. By distributing this large collection of samples we want to enable other security professionals to gain insights into the tactics, techniques, and procedures (TTPs) used by threat actors to develop and deploy these implants.

Before starting on the implant generation process, we had to take several preparatory steps to ensure that the process runs smoothly and effectively. Firstly, we had to understand the set of possible options and their respective values for each C2 framework. Secondly, it was necessary to understand how to interact with various C2 frameworks to generate the implant. Interacting with various C2 frameworks can be a challenging task due to their differing interfaces. While some frameworks, like Sliver, offer user-friendly command-line interfaces with multiple options, others, such as Cobalt Strike, can only be interacted with via their proprietary mechanisms. Brute Ratel and Covenant proved to be the most challenging, as they required us to reverse engineer their communication protocols.

Specifically, Brute Ratel's protocol requires the client to authorize twice: the first time by sending the login and the password via an HTTP POST request, and then a second time with the token received from the first authorization attempt, sent via a newly established WebSocket channel. After successful authorization, the WebSocket connection is used for client-server communication where the client sends commands (e.g. 'create a badger profile with the following parameters') and the server replies with status codes and the additional data (e.g. a Base64-encoded payload) that the client might have requested. Both channels (the initial POST request and the WebSocket connection) are JSON-based and use HTTPS as a transport layer.

Covenant, on the other hand, has a documented set of APIs that include functions to create implants. The APIs are based on JSON and use HTTPS as a transport layer. While most of the functions indeed generate implants, some of them (e.g. the function to generate a .NET executable implant) return no payload with the standard reply. The payload is generated but never returned, because that functionality is not implemented. To overcome this limitation the code of the framework was amended to store the generated payload on disk.

Once the options for each implant type had been identified, we created a custom domain-specific language to express them in a format that can be easily consumed by the implant generation process. Careful consideration and planning was required to ensure that the process was efficient and scalable. Finally, we implemented an algorithm that, given the grammar of one of the implant configurations expressed with our DSL, can generate random configurations that are consistent with the grammar. These configurations are then used to generate the implants. This algorithm can be used to generate a large number of implant variations, each tailored to specific target environments, and can be used to test the effectiveness of various detection and defence strategies.

In the next section we will describe the high-level architecture of our C2F2 framework.

C2F2: A C2 FRAMEWORK FRAMEWORK

We designed our framework based on two key properties of the problem we needed to solve. Firstly, generating a single implant can be a time-consuming process that may take several minutes to complete due to the complexity of the steps involved, such as, in the case of Cobalt Strike, creating a Malleable C2 profile, starting a Cobalt Strike server, using Aggressor Scripts to generate the implant binary, and waiting for the result to be produced. Secondly, the process of generating one implant is independent of generating another, meaning that multiple implants can be generated simultaneously in parallel, which can significantly increase the efficiency of the process. To address these two factors, we ensured that our framework was capable of handling asynchronous long-lasting jobs and designed it to be easily parallelizable.

Our infrastructure consists of four key components that work together to facilitate the process of generating implants:

- Generator: This component retrieves the correct grammar based on the specified C2 framework type and generates a random implant configuration. The configuration is then stored in the designated storage backend for future use.
- Submitter: Once an implant configuration is available, the submitter creates a job and sends it to the appropriate queue, based on the configuration type.
- Receiver: This component pulls jobs from the queue and sets up the worker to generate the corresponding implant. Once the job is completed, the receiver collects the result.
- Worker: Each worker is specialized in generating implants for a specific C2 framework. Given an implant configuration, the worker generates the corresponding implant. By dividing the workload across multiple workers, our infrastructure is able to generate multiple implants in parallel, significantly reducing the time required to generate large datasets. Overall, these components work seamlessly together to automate the implant generation process and improve the efficiency of our system.

As shown in Figure 10, the user begins by specifying the C2 framework type and the number of implants to be generated, initiating the process. The generator component then retrieves the appropriate grammar and generates a random implant configuration, which is then stored in the chosen storage backend. The submitter retrieves the generated configuration file, validates it, and creates a job and sends it to the appropriate queue based on the configuration type, kicking off the process of generating the implant. The receiver pulls jobs from the queue and sets up the Docker container running the appropriate specialized worker. The worker then receives the implant configuration and, using the C2 framework backend, generates a binary. The receiver collects the result and stores it in the designated storage backend. This process is repeated until the

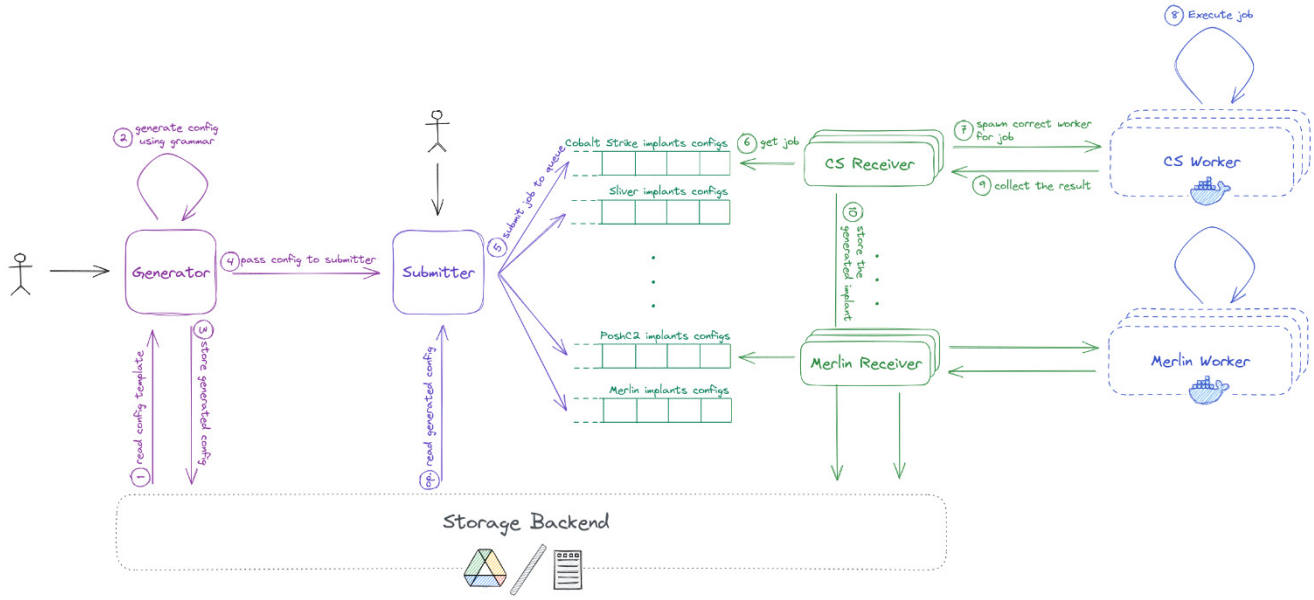

Figure 10: C2F2 architecture.

specified number of implants has been generated. Additionally, the user has the option to provide a custom implant configuration, skipping the configuration generation phase. In this case, the submitter creates a job and sends it to the appropriate queue based on the configuration type, and the process continues as usual. This workflow ensures that the implant generation process is streamlined and allows for easy customization.

The datasets

Our framework produced 9,950 different implants across the 10 frameworks previously described. In addition to the datasets produced by the C2F2 infrastructure, we also collected 1,980 benign samples (collected from various operating systems' basic installations) and 1,026 malicious ones.

The dataset is composed entirely of implants in binary format. Table 2 shows the binary formats produced by the various C2 frameworks.

Name	PE (Windows)	ELF (Linux)	Mach-O (OSX)
Cobalt Strike	X		
Metasploit	X		
Sliver	X		
Brute Ratel	X	X	
Covenant	X		
Merlin	X		
Shad0w	X		
Empire	X		
PoshC2	X		
Godoh	X	X	X

Table 2: List of binary formats.

LEARNING TO LOVE THE RAT

Given the datasets generated by our C2F2 framework, we developed a detection approach based on machine-learning techniques.

More precisely, we designed a deep-learning model that operates on raw bytes extracted from the PE binaries. The goal of this model is to accurately identify whether a binary is simply malicious, malicious and generated by a C2 framework, or benign. We achieve this goal by performing multi-label classification. Previous research has yielded promising results by using raw bytes from binaries as input features for deep-learning models with the task of malware classification. However,

the nature of deep neural network models is such that multiple issues need to be addressed. Firstly, the sheer number of bytes present in a binary prohibits the use of each byte as a token. Secondly, there exists high positional variation in executable files (see for example [44]). While the contents of a PE binary can be rearranged in almost any arbitrary order, only the MS-DOS header is constant. This header ends with a pointer to the beginning of the PE Header, which contains pointers to all other contents of the binary. This allows macro-reorganization of byte contents without changing their context. This spatial restructuring can make it challenging to classify binaries accurately.

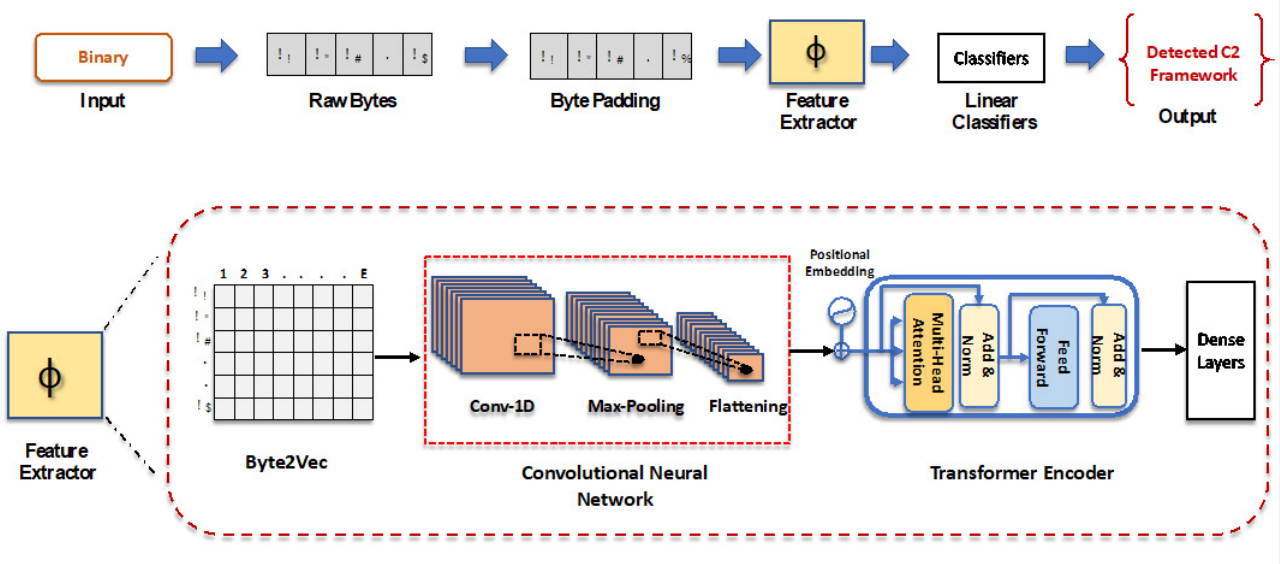

Figure 11: Machine-learning pipeline for the classification of C2 implants.

To overcome these challenges, our deep-learning model performs convolutions on the feature representation of a fixed number of bytes. Raw bytes first undergo processing in the embedding layer, which maps each byte to a fixed-length feature vector. Following this step, 1D convolutions are performed on the output of the embedding layer. The convolutional activations are combined with global max-pooling to enable the model to produce its activation, regardless of the location of the detected features. Jointly training the embedding with the convolution allows the shallow network to activate for a broader range of input patterns and provides robustness against minor alterations in byte values. The output of the convolutional neural network layer is a feature map of much lower dimension than the total bytes in the binary. This feature map is then processed by a transformer encoder to learn sequential features, and the resulting output is processed together by the dense layers before classification. This proposed model can accurately classify malware and implants (malware generated by different C2 frameworks) even when the size of the binary is large, or when the byte contents are rearranged or slightly altered. The overall structure of our ML classification pipeline can be seen in Figure 11.

The results of our experiment show that the system is able to classify implants belonging to the 10 C2 frameworks with an accuracy that varies between 99.1% and 99.9%. In addition, the system was able to identify both generic malicious samples and benign samples with an accuracy of 98%. In every classification task, the number of false positives was negligible.

CONCLUSIONS

C2 frameworks are used by cybercriminals to establish a bridgehead in enterprise networks. The generative, polymorphic nature of the implant generation process is used to evade detection but can also be leveraged to generate large datasets for machine learning.

To this end, we built C2F2, a C2 framework analysis framework that supports the large-scale generation of implants. Using C2F2, we have demonstrated how the generated implant datasets can be used as input to a machine-learning pipeline, with the goal of creating a solution that is able to detect implants and classify them with very high precision.

As part of this research, we also release C2F2 on *GitHub*, along with the dataset of generated implants that have been used to train the machine-learning model.

REFERENCES

[1] SANS. Adversary Emulation and the C2 Matrix. https://www.sans.org/webcasts/adversary-emulation-c2-matrix-113500/.

[2] SANS Slingshot C2 Matrix VM. https://howto.thec2matrix.com/slingshot-c2-matrix-edition.

[3] https://www.virustotal.com/gui/file/aabe4b60c5a7c64e284057193b369eb2b078dbdae945092179f224240352e47e.

[4] https://www.virustotal.com/gui/file/591c2cd3a9b902a182fbf05bf5423cae17e3e6874c0d2e09107e914d86f39780.

[5] https://www.virustotal.com/gui/file/56a53682084c46813a5157d73d7917100c9979b67e94b05c1b3244469e7ee07a.

[6] https://github.com/Freakboy/CobaltStrike.

[7] Cobalt Strike. https://www.cobaltstrike.com/.

[8] C2concealer. https://github.com/RedSiege/C2concealer.

[9] Cobalt Strike User Guide. Aggressor Script. https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/agressor_script.htm.

[10] Jazi, H.; Segura, J. Multi-stage APT attack drops Cobalt Strike using Malleable C2 feature. Malwarebytes. 17 June 2020. https://www.malwarebytes.com/blog/news/2020/06/multi-stage-apt-attack-drops-cobalt-strike-using-malleable-c2-feature.

[11] Lakshmanan, R. LockBit Ransomware Abuses Windows Defender to Deploy Cobalt Strike Payload. The Hacker News. 2 August 2022. https://thehackernews.com/2022/08/lockbit-ransomware-abuses-windows.html.

[12] Google Cloud. Making Cobalt Strike harder for threat actors to abuse. 18 November 2022. https://cloud.google.com/blog/products/identity-security/making-cobalt-strike-harder-for-threat-actors-to-abuse.

[13] The Metasploit Framework. https://www.metasploit.com/.

[14] Sliver. https://github.com/BishopFox/sliver.

[15] AhnLab ASEC. Sliver Malware With BYOVD Distributed Through Sunlogin Vulnerability Exploitations. 6 February 2023. https://asec.ahnlab.com/en/47088/.

[16] Microsoft. Looking for the 'Sliver' lining: Hunting for emerging command-and-control frameworks. 24 August 2022. https://www.microsoft.com/en-us/security/blog/2022/08/24/looking-for-the-sliver-lining-hunting-for-emerging-command-and-control-frameworks/.

[17] Cybereason. Sliver C2 Leveraged by Many Threat Actors. 19 January 2023. https://www.cybereason.com/blog/sliver-c2-leveraged-by-many-threat-actors.

[18] Proofpoint. Security Brief: TA551 Uses 'SLIVER' Red Team Tool in New Activity. 20 October 2021. https://www.proofpoint.com/us/blog/security-briefs/ta551-uses-sliver-red-team-tool-new-activity.

[19] Brute Ratel. https://bruteratel.com/.

[20] SOCRadar. Brute Ratel Utilized By Threat Actors In New Ransomware Operations. 7 July 2022. https://socradar.io/brute-ratel-utilized-by-threat-actors-in-new-ransomware-operations/.

[21] Harbison, M.; Renals, P. When Pentest Tools Go Brutal: Red-Teaming Tool Being Abused by Malicious Actors. Unit 42 Palo Alto Networks. 5 July 2022. https://unit42.paloaltonetworks.com/brute-ratel-c4-tool/.

[22] Godoh. https://github.com/sensepost/godoh.

[23] DNS Queries over HTTPS (DoH). https://www.rfc-editor.org/rfc/rfc8484.

[24] Shad0w. https://github.com/bats3c/shad0w.

[25] SANS. Slingshot Linux Distribution. https://www.sans.org/tools/slingshot/.

[26] Secureworks. Phases of a Post-Intrusion Ransomware Attack. 28 July 2021. https://www.secureworks.com/research/phases-of-a-post-intrusion-ransomware-attack.

[27] Empire. https://github.com/BC-SECURITY/Empire.

[28] BC Security. https://www.bc-security.org/.

[29] Donut. https://github.com/TheWover/donut.

[30] CISA. Cybersecurity Advisory. #StopRansomware: Vice Society. https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-249a-0.

[31] Mandiant. FIN12 group profile: FIN12 prioritizes speed to deploy ransomware against high-value targets. https://www.mandiant.com/sites/default/files/2021-10/fin12-group-profile.pdf.

[32] Merlin. https://github.com/Ne0nd0g/merlin.

[33] RFC 7518: Key Encryption with PBES2. https://www.rfc-editor.org/rfc/rfc7518#section-4.8.

[34] The OPAQUE Asymmetric PAKE Protocol. https://datatracker.ietf.org/doc/html/draft-krawczyk-cfrg-opaque-00.

[35] Bitdefender. Cyber-Espionage in the Middle East: Investigating a New BackdoorDiplomacy Threat Actor Campaign. https://www.bitdefender.com/files/News/CaseStudies/study/426/Bitdefender-PR-Whitepaper-BackdoorDiplomacy-creat6507-en-EN.pdf.

[36] PoshC2. https://github.com/nettitude/PoshC2.

[37] PoshC2 (specifically as used by APT33). https://github.com/jeFF0Falltrades/IoCs/blob/master/APT/poshc2_apt_33.md.

[38] Checkpoint Research. DangerousSavanna: Two-year long campaign targets financial institutions in French-speaking Africa. 6 September 2022. https://research.checkpoint.com/2022/dangeroussavanna-two-year-long-campaign-targets-financial-institutions-in-french-speaking-africa/.

[39] Bone, R. Detecting PoshC2 – Indicators of Compromise. LRQA Nettitude Labs. 17 June 2020. https://labs.nettitude.com/blog/detecting-poshc2-indicators-of-compromise/.

[40] Censys. Russian Ransomware C2 Network Discovered in Censys Data. 18 July 2022. https://5851803.fs1.hubspotusercontent-na1.net/hubfs/5851803/Russian%20Ransomware%20C2%20Network%20Discovered%20in%20Censys%20Data.pdf.

[41] Covenant. https://github.com/cobbr/Covenant.

[42] Landwerth, I. Introducing .NET Core. Microsoft .NET Blog. 4 December 2014. https://devblogs.microsoft.com/dotnet/introducing-net-core/.

[43] Symantec. Sophisticated Espionage Group Turns Attention to Telecom Providers in South Asia. 19 May 2020. https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/greenbug-espionage-telco-south-asia.

[44] Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C. Malware Detection by Eating a Whole EXE. https://arxiv.org/pdf/1710.09435.pdf.