Design Document

Team 19 - Offensive Security Orchestration

Draft #1 | October 12, 2018

Table of Contents

Introduction

Problem and Project Statement

Our project aims to develop a security orchestration platform for our client which will allow them to conduct red team engagements in a stealthy and efficient manner. Since our client often uses widely deployed and standardized tools during red team engagements they frequently run into issues with the tools they use being flagged by network security teams. By developing a custom implant which integrates into the security orchestration platform we will be able to bypass the tools which flag on more common implants. The other issue our client encounters is that developing red team processes is a manual and time consuming endeavor. This is a result of needing to test implants against multiple endpoint detection solutions to see if the implant will be discovered. To address this issue the security orchestration platform will automate the deployment of implants so that our client can rapidly develop and test different payloads.

Operational Environment

The functional requirements of our system are composed of two primary components that work both independently and together but supply their own unique functionality. Those components are the malware (“Bot”) and the command and control frontend (“C2”).

Intended Users and Uses

The intended users of our project will be the company red team. We will have various levels of security and user authentication to ensure there is no misconduct with the tool. There will be a history log of all actions done along with Admin and Regular accounts. They will be able to utilize our project to conduct engagements in a stealthy manner. This tool will be able to bypass security tools that commonly flag regular implants. Also with our automated deployment system, they will be able to rapidly test multiple different payloads.

Assumptions and Limitations

The project does have a cost of needing AWS credits so we are under the assumption that our company client will be paying for the costs. AWS access stands for Amazon Web Services which is a cloud hosting system that’ll allow us to host our server/application as well as utilize up to 8 possible VMs.

Another assumption is that we will be able to complete all of the deliverables on time. As we reviewed all of them we found out that we had two very difficult deliverables. Those were, getting results from the EDR (endpoint detection and response) and the In-App malware builder

that'll use the web app to compile malware with variables. Being able to complete these on time isn't an easy task due to the difficulty they present.

The project and corresponding system will comply with the following limitations:

- The C2 will use ReactJS framework for the user interface SPA
- The C2 backend will use Django
- The C2 will communicate to the bots via REST APIs and be considered "RESTful"
- The EDR solution is restricted to whatever endpoint protection services we are able to get versions or trials for. Some proprietary/commercial software is hard to obtain legal licenses for, such as trials.

Expected End Product and Deliverables

Deliverables for the Implant Team (C# Back-End)

- Domain Fronting using Amazon Cloudfront
- Macro to deliver payload malware through automated phishing
- Scheduled task persistence while remaining stealthy
- Bypassing endpoint detection and response solutions

Deliverables for the Command and Control Team (Web App Front-End)

- User authentication with multiple levels of access
- Sockets for bi-directional communication between implants and controller
- Encrypted communication (HTTPS)
- Logging of actions taken by every user
- Admin action page for actions such as user management
- Dockerize application to standardize deployment
- Creation of help pages for common actions a user will take
- In-App Malware tester that allows users to quickly test different payloads against EDR solutions

Specifications and Analysis

Proposed Design

Our team has met extensively with the client to lay out all the desired and necessary requirements. Those requirements are as follows and are separated by our two subteams:

Command-And-Control Requirements

- User authentication with multiple levels of access

- Sockets for bi-directional communication between implants and controller
- Encrypted communication (HTTPS)
- Logging of actions taken by every user
- Admin action page for actions such as user management
- Dockerize application to standardize deployment
- Creation of help pages for common actions a user will take
- In-App Malware tester that allows users to quickly test different payloads against EDR solutions

Implant Requirements

- Domain Fronting using Amazon Cloudfront
- Macro to deliver payload malware through automated phishing
- Scheduled task persistence while remaining stealthy
- Bypassing endpoint detection and response solutions

Design of Command-And-Control Element

Our Command-And-Control system, or C2 for short, will be a single-page web application with a dedicated frontend and backend. We have decided on the following technologies and frameworks:

Frontend
- ReactJS UI framework
- Semantic UI CSS framework

Backend
- Django Python Web framework
- Django REST Framework (for APIs)

The frontend and backend will communicate almost exclusively via HTTP REST APIs, provided by the Django REST Framework (DRF), a wholly supported, financed, and dedicated framework to providing stable RESTful Application Programmer Interfaces (APIs). The backend database will be SQLite to begin with, and if resources demand it, we will look at moving to a dedicated database service such as MariaDB or Postgres.

We will utilize Docker for easy deployment of this application in terms of microservices. This will allow the client to spin up an instance of the C2 anywhere (their laptop, a dedicated server, in the cloud, etc.), and without any initial configuration needed.

Design of Implant Element

The implant element will be written using the C# programing language. Amazon Cloudfront, part of Amazon Web Services (AWS) will be used to implement Domain Fronting.

Design Analysis

At this point in time, our code base consists of a simple reconnaissance implant in C# that was provided by the client. The implant can retrieve and send data to the C2 server as well as run commands on its infected host. In addition to the implant, we have a working model of the C2 application, previously written by one of our team members. This barebones application utilizes our existing technologies and frameworks.

Our team has met with the client to discuss what they would like completed during the course of this project. So far we have, in depth, defined the scope of the project and desired outcomes.

Besides defining scope, our team has been meticulously researching methods of implementation for the each of the demanding requirements for this project. Our C2 and implant modules require significant low-level knowledge of the Windows Operating System as well as web application development, so at this time everyone is familiarizing themselves with the technologies they will be using before implementation. Any missteps or wrong paths taken during this project can severely hinder progress down the road with this project, so research is being taken very seriously in these initial stages.

Research is beginning to pay off - our team is having more in depth discussions about proper ways to implement certain features, and members are now understanding the technologies they will be working with.

Our thoughts on the design are as follow: we believe we have a lot of highly-technical and challenging work ahead of us, but we have carefully selected the proper technologies and implementation routes to ensure that we provide our client with a stable and professional product that they can use in their Red Team Engagements.

Testing and Implementation

Interface Specifications

The initial loader will be distributed through a word macro and it will therefore need to interface directly with Microsoft Word. From there the loader will download the implant and write it to C:\Windows\Temp. Communication between the command and control server will be handled with HTTP/S and will allow for the implant to receive commands from the control server using REST API's.

Hardware and Software

The loader provides a means of easily installing our implant on a server without needing to package the whole thing in a Word Macro, or distribute binaries directly. This allows for the potential to do some reconnaissance of the infected host to determine whether or not it is a place that we want to install to. However we will need to do some testing to determine what kinds of reconnaissance we want to perform, and how we will get the loader working with our fronted domain.

In order to test the loader we have determined that we will follow a simple procedure to determine whether or not we have fallen prey to a number of common pitfalls. These pitfalls include, improper generation of the Word Macro, misconfigured C2 server ip address, and not using our fronted domain. Our procedure for testing these pitfalls is detailed in the process section below.

Due to the limited functionality our implant currently has we have determined that it makes more sense to manually test whether or not all of the functionality exists. For instance we only have a few different commands that can be run and so it is easy enough to test whether or not we can upload/download a file, send shell commands, and receive responses all through the Command & Control server.

While testing the communication between the implant and the Command and Control server we have found that the most useful tool for the job is Wireshark. Using Wireshark we can see whether or not packets are going through the fronted domain, as well as seeing if the proper REST API's are being used to send data.

Initial testing of the Command & Control server itself has been done mainly using Docker. This is because the entire application has been built so that way we can use docker-compose to setup the entire server automatically so that our client can easily setup, and tear down the server as necessary for each engagement.

As of now we have only completed minimal testing for our endpoint detection system. However the process will be very similar to the Command & Control server because we will be again be using Cuckoo Sandbox within a Docker container so that we can easily monitor whether or not the server is deployed properly.

One of the major implementation challenges we will face is the communication between the Cuckoo Sandbox, and the Command & Control server. In an ideal world we would like them to be built into the same web application so that our client will only have one place that they need to go in order to see all of the endpoints that will be affected by running certain payloads. Therefore we need to find a way to take the web application provided by Cuckoo, and combine it with our web application for our Command & Control server.

Process

We will first begin the design and testing of the loader. We need to start with this because the loader is what starts the entire process off, and without it we have no way of getting the implant onto the victim machine. Our procedure to test the loader is mostly manual due to the nature of our project not being something that can easily be tested automatically. However we do use some different software tools to speed up the process and quickly determine issues as they arise. One of these tools is Microsoft Word itself because the easiest way to determine if a macro is functional is to simple run it within Word. Next, we can use Wireshark to rapidly determine what issues are arising with the loader. These issues include whether or not the loader is reaching out to the correct IP address, and also ensuring that the fronted domain is being used to help hide our tracks on the network.

We will be using a similar testing process while developing the implant. However since the implant includes more functionality than the loader we are including a few more tools to our process. These tools are Windows tools Process Monitor, and Process Explorer. They allow us to view the whether or not our implant is being ran, while also seeing under what context it is running. Therefore our process for testing the implant is simple. First we will view whether or not the implant is properly downloaded and installed into C:/Windows/Temp by the loader. If not then we will go back and go through the process for testing the loader again to see what is going wrong. Next, we will test whether or not the implant is being run by Windows Script Host. We can test this using Process Explorer and seeing if there is an instant of Windows Script Host being ran under our current users context.

Finally our process for testing the C2 server is able to be automated however we have not yet determined if that will be necessary. Due to the entire server being deployed using docker we believe that the portion of the testing that could be automated will all be handled by docker. For example determining if the correct ports have been opened, if the container has internet, if implants can reach the C2 server. Then the more complex portions of our environment can be tested easily by one person running a few commands for an implant and seeing whether or not

data is properly returned. Since the data will be directly output onto the web application it makes more sense to have a person directly inputting the commands rather than have a automated program run them and then pull the output from the web application.

Results

This project is a two-semester project and is still in progress. Therefore we do not have conclusive results however we do have preliminary calculations, estimations, and details from our research and time spent developing this project so far. The current results we have obtained can be found in the Design Analysis and Testing and Implementation sections of this document.

Closing Material

Conclusion

So far, our team has done extensive research, discussion, and some amount of testing various methods for the implementation of our deliverables. This process allowed verification that our different ideas were technically feasible, which was extremely important for the implant element, as it requires less frequently used techniques.

Our overall goals are divided into two main categories: Command and Control, and Implant. Goals in the Command and Control category consist of requirements that result in making a webapp more secure, more user-friendly, have more functions, and be easier to set up. Goals in the Implant team consist of setting up domain fronting, creating a malicious macro for Word, allowing tasks to persist stealthily, and bypassing endpoint detection and response.

To achieve these goals, the first step in our plan was to split up the team into two subteams to work on either category of the goals. This was chosen to be done as the requirements needed for each can be done without much reliance on the other one. This allows us to get work done in parallel, and the separation into subteams formalizes this. For the Command and Control team, our plan is to start with Docker before working on the other requirements. This is because getting Docker set up correctly will make the process of working on everything else go faster. For the implant team, our plan is to start with the malware loader, as it is a prerequisite to work on the other requirements.

References

1. Caparas, Marianne, and Nick Schonning. “Overview of Endpoint Detection and Response Capabilities.” Capabilities | Microsoft Docs, 2 Sept. 2018, docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-atp/overview-endpoint-detection-response.
2. Christensen, Lee. “EmpireProject/PSInject.” GitHub, 11 Apr. 2017, github.com/EmpireProject/PSInject.
3. Fielding, R., et al. “Hypertext Transfer Protocol -- HTTP/1.1.” IETF Tools, June 1999, tools.ietf.org/html/rfc2616.html.
4. Guarnieri, Claudio. “Cuckoo Sandbox Book¶.” Cuckoo Sandbox Book - Cuckoo Sandbox v2.0.6 Book, The Honeynet Project, 20 Sept. 2017, docs.cuckoosandbox.org/en/latest/.
5. Limanowski, Daniel. “b1tst0rm/Django-React-Boilerplate.” GitHub, 8 July 2018, github.com/b1tst0rm/django-react-boilerplate/blob/master/README.md.
6. Nelson, Matt. “WSH Injection: A Case Study.” enigma0x3, 3 Aug. 2017, enigma0x3.net/2017/08/03/wsh-injection-a-case-study/.