eEye® Digital Security

Payload Anatomy and Future Mutations

Riley Hassell
rhassell@eeye.com

What is a Payload?

Traditional payloads are written in assembly language and compiled to their machine code counterpart. They contain a series of automated tasks that help an attacker to gain a more comprehensive level of control over his target.

Common payloads used in exploits and by worms perform a simple set of tasks that can be easily detected and prevented. As security systems advance over the next several years, attackers will be forced to develop more sophisticated payloads that permit them to silently bypass new checks and restrictions.

Payload Anatomy and Future Mutations will provide a general overview of techniques used to construct and deliver a new generation of payloads. The research reviewed during this presentation is intended to assist the security community with predicting the future of payload technologies.

eEye® Digital Security

Overview

- Placement
  - Avoiding restrictions
- Stealth
  - Evading detection
- Reliability
  - Payload hardening
- Optimization
  - Payload optimization
- Payloads
  - Fakeload and RIO

eEye® Digital Security

Placement

During the initial injection of the payload into the virtual address space of the target application, the attacker may run into a series of restrictions. By utilizing features and protocol extensions supported by the target application, payloads can be injected so that they mimic the normal behavior of the application. This allows the payload to navigate around many, and sometimes all, of these restrictions.

- **Bypassing size restrictions**
  - While it may seem that the placement area is very limited, there is usually more space to work with than meets the eye.
- **Bypassing byte value restrictions**
  - It is possible to navigate around strong filtering such as islower() checks by simply relocating the payload elsewhere.
- **Navigating to control structures**
  - While a security hole may be difficult to exploit reliably, altering session characteristics can be used to reach a higher level of reliability.

eEye® Digital Security

Stealth

Payloads may contain support to evade various normalization and signature based security systems that they encounter along the path to the target application. This section provides an overview of common techniques and technologies used to evade detection.

- **Utilizing System Resources**
  - Exploits may utilize the resources provided by the target to mimic normal application behavior.
- **Morphing Engines**
  - Powerful engines designed to “morph” a payload so that it can bypass signature tests.

eEye® Digital Security

Utilizing System Resources

Exploits may fully utilize the target's protocol support and added features to disguise their payloads, including encoding, compression, and encryption.

- **Encryption**
  - If the target offers any form of encryption, the payload may use that medium instead of the clear text transport medium, and will most likely sneak by the majority of IDS systems.
    - File format vulnerabilities
- **Compression**
  - If the target supports any transport compression, the payload may be compressed in the stream and decompressed by the server before the vulnerable condition is triggered.
    - File format vulnerabilities
    - Media-based protocols server vulnerabilities
- **Encoding**
  - Many protocol server implementations offer encoding schemes to support data types that require more than data. Simple authorization mechanisms that do not use encryption will most likely use simple encoding schemes.
    - UTF
    - Base64

eEye® Digital Security

Morphing Engines

Worms and virii have used morphing engines for decades to evade signature-based Anti Virus systems. The same technologies can be used to evade other simple signature-based security systems, such as Intrusion Detection Systems. The virus writers of yesterday are the worm writers of today -- their technologies are highly advanced and their research provides valuable insight into security systems.

- **Polymorphism**

  Polymorphism engines (from the Greek meaning "having multiple forms") have been around for quite a while in the VX scene. Virus writers initially implemented polymorphism to bypass simple signature matching AV systems.

- **Metamorphism**

  "Metamorphism": the process by which the material of rock masses has been more or less recrystallized by heat, pressure, etc., as in the change of sedimentary limestone to marble. To a virus writer this portrays a more intense and technically adept level of polymorphism.

eEye® Digital Security

Reliability

To increase the chances of a payload executing successfully when in an unstable or unreliable machine environment, various technologies may be utilized to “harden” the payload and improve the independency of its surroundings.

- **Dynamic Loading**
  - Engines that take advantage of the target’s linking and loading schemes to permit the attacker access to the same API suite available to the target.
- **Fault Handling**
  - The target process can become very unstable depending on the type of vulnerability being exploited. To handle these type of conditions, routines may be wrapped with SEH handlers and/or the default exception filter may be replaced.

eEye® Digital Security

Dynamic Loading

When an exploit is constructed for a vulnerability affecting multiple versions of a software product, the payload must be able to operate regardless of the various differences in the virtual address space of the target process. Dynamic Loader Engines, nicknamed (RVA), are available to handle the differences in library and executable versions within the target.

- **Based Loader Engines**
  - Import Loaders
  - Export Loaders
- **Baseless Modifications**
  - Baseless Import Loading
  - Baseless Export Loading

eEye® Digital Security

Fault Handling

The target process can become very unstable depending on the type of vulnerability being exploited. To handle these type of conditions, payload routines may be wrapped with SEH handlers, and the default exception filter may be replaced so any fault can be gracefully handled.

- **Initializing**

```
SetUnhandledExceptionFilter((LPTOP_LEVEL_EXCEPTION_FILTER)handler)
```

- **The Handlers**

```
void termination_handler(void)
{
        ExitThread();
}

void suspension_handler(void)
{
        SuspendThread(GetCurrentThread());
}
```


eEye® Digital Security

Optimization

Virus writers, worm writers, and exploit designers will agree that payload design is an art form. While Van Gogh and Dali use shapes and palette schemes, these digital artists of the 21st century use size, speed, stealth and an unforeseen level of complexity to convey their abilities.

- **Smaller instructions**
- **Low cycle instructions**
- **Instruction pairing**
- **Using added chip features**
- **Stack tricks**

eEye® Digital Security

Fakeload

"Loading" is the process of creating an executable image in memory from an application on disk so the program can run. When a program is loaded, a new process is created, necessary resources are allocated, and a new virtual address space is initialized. “Fakeloading” is very similar to loading, but the PE image is taken from memory and an existing virtual address space is used. The Fakeload engine performs the following steps to successfully Fakeload an image:

- The PE image is downloaded across an SSL connection directly into memory.
- The downloaded image is now treated as a PE file in memory.
- The Fakeload engine examines the section table, allocates and copies the sections into the relevant RVA offsets in memory.
- The import table is examined, necessary libraries are loaded, and function entry points are stored.
- After importing, relocation must be performed. The relocation section is reviewed and each section chunk is fixed so they align politely with the new base address.
- The Fakeload engine now directs execution into the entry point specified in the Fakeloaded PE image.

eEye® Digital Security

Fakeload Benefits

- The "Fakeloading" concept permits an attacker to execute a remote application using the existing virtual address space and process of his target.
- Since there is no file opened on disk, no process created, no thread created, or any other activity that would notify the operating system that the process has changed, the majority of local security systems are evaded.
- The payload can be built using a variety of programming languages, eliminating the time-consuming process of developing complex payloads in assembly language.
- All the applications available to protect PE executables can be used on developed fakeload modules. For example: packers, crypters, anti-debugging, etc.

eEye® Digital Security

RIO

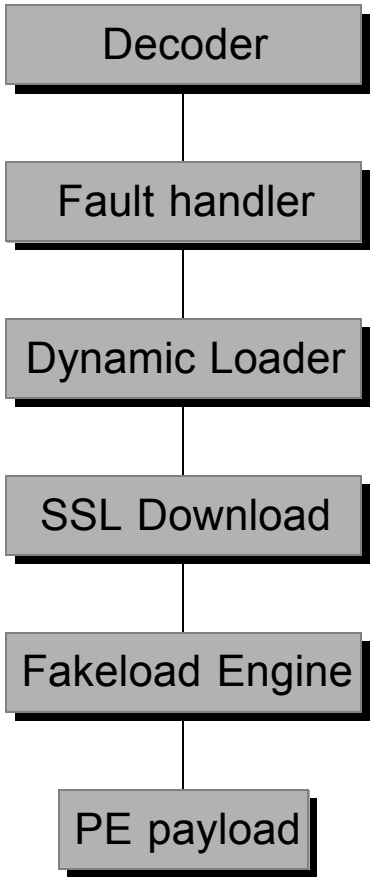
The Radar-Intercept Officer, “RIO”, is the nickname of the person in the back seat of the F14 Tomcat that guides the Naval Aviator to the optimum course and speed to intercept hostile aircraft. RIO’s also operate the complex navigation, sensor, and weapons systems onboard.

The RIO payload was the first module designed for use with the Fakeloader engine.

- RIO initializes active network interfaces and examines network traffic for any data relevant to a signature rule.
- Once RIO has identified the relevant data, a RIO capture structure is created containing source IP, destination IP, source port, destination port, and data. RIO will continue to create capture blocks until a set maximum size is reached.
- Once the threshold is met, RIO will unload and await the attacker’s request for the captured information. RIO can also be configured to deliver the data to a necessary destination, using a configured medium.

eEye® Digital Security

Fakeload and RIO



- **This decoder allows for navigation around the majority of input filtering schemes.**
- **A fault handler is installed here, so if at any time the payload encounters severe problems it can cleanly exit the process.**
- **The "Baseless" Import Loader loads the libraries and functions used by the following components.**
- **This module downloads a remote file into the virtual address space of the target, using an SSL encrypted connection.**
- **The Fakeload engine, as described before, simply loads a PE payload into the virtual address space of the target process.**
- **A PE executable, written in "C", and previously built. RIO is an example PE payload.**

eEye® Digital Security

The Future…

- Multiple architecture engines
- Multiple stage payload injection
- Passive payload injection
- Task-based payloads, multi-tier designs
- Privilege escalating engines

eEye Digital Security Research Department
http://www.eeye.com/research/

eEye® Digital Security