A NEW VILLAIN: INVESTIGATING STEGANOGRAPHY IN SOURCE ENGINE BASED VIDEO GAMES

Christopher Hale
Department of Computer Science
Sam Houston State University
Huntsville, Texas
chris.hale@shsu.edu

Lei Chen
Department of Computer Science
Sam Houston State University
Huntsville, Texas
chen@shsu.edu

Qingzhong Liu
Department of Computer Science
Sam Houston State University
Huntsville, Texas
liu@shsu.edu

***Abstract*—In an ever expanding field such as computer and digital forensics, new threats to data privacy and legality are presented daily. As such, new methods for hiding and securing data need to be created. Using steganography to hide data within video game files presents a solution to this problem. In response to this new method of data obfuscation, investigators need methods to recover specific data as it may be used to perform illegal activities. This paper demonstrates the widespread impact of this activity and shows how this problem is present in the real world. Our research also details methods to perform both of these tasks: hiding and recovery data from video game files that utilize the Source gaming engine.**

Keywords-steganography, Steam, Source, video games, digital forensics, investigation, Hammer

I. INTRODUCTION

With the growing amount of information and responsibility placed on computer systems comes an increased threat of misuse and abuse of these systems. Safeguards must be developed in conjunction with technology in order to ensure its safety and keep threats in check. In a most recent report by Symantec, they found over 286 million unique malware variations in circulation. The report also shows a 93% increase in web attacks, a 42% increase in mobile device vulnerabilities, and 6,253 new software vulnerabilities. These numbers represent the largest yearly increase in the fifteen years that this study has been conducted [1]. All of these numbers represent the inherent threats that computers and electronic devices bring to their users.

As the threat of computer crime grows, so does the number of avenues which criminals may use to conduct illegal and potentially damaging activities. One of the newest and often overlooked threats comes from a seemingly innocuous source: video games. In the not too distant past, creating a video game was a relatively small venture. A team of one or two individuals could create, publish, and release a game on their own. Since its humble beginnings, the art of video game development has become an enormous commercial success. With over 72% of all American households playing video games and $4.9 billion in revenue [2][3], this industry is booming more now than ever before. As video game business continues to grow and develop, the potential for exploiting these services proportionately increases. Video games vulnerabilities are not often seen as serious security threats by individuals and security professionals. This paper outlines several of these threats and how they can be used to transmit illegal data and conduct potentially illegal activities. It also demonstrates how investigators can respond to these threats in order to combat this emerging phenomenon in computer crime.

This paper is organized as follows. In Section II we introduce the Source Engine, one of the most popular game engines, Steam, a powerful game integration and management tool, and Hammer, an excellent tool for creating virtual environment in video games. In Section III we look at various ways of hiding data in video games using the above tools. Section IV discusses methods for investigators to detect hidden data in game files and environments. We draw conclusion and lay out future work in Section V.

II. THE SOURCE GAMING ENGINE

This paper primarily focuses on threats presented by the Source gaming engine. This engine is owned and developed by the Valve Corporation. Due to its extremely large user base and commercial popularity, it is one of the most popular in the world of gaming.

A. The Valve Corporation: Creators of the Source Engine

Kirkland, Washington, in the year of 1996, Valve was founded by Gabe Newell and Mike Harrington, two previous Microsoft Windows developers. The company started to create innovative and groundbreaking new video games. Valve initially worked on developing several small projects through the next two years, eventually abandoning these plans and focusing their resources on their first commercial release: Half-Life. Since its release in 1998, Half-Life received over fifty Game-of-the-Year Awards as well as being heralded as "one of the best games ever" [4]. Following the commercial success of Half Life, Valve released its next successful game: Counter-Strike. Currently, Counter Strike 1.6 is the most widely played online video game in the world with the exception of Massively Multiplayer Online Role Playing Games [5]. In September 2003, Steam was released as a tool to seamlessly integrate updates into the Counter Strike franchise. It gradually saw greater integration into Counter Strike and all Valve game releases. Since the release of Half-Life 2 in 2004, Valve has released a number of titles, each using an improved and altered version of the Source game engine. Many of these games have achieved immense commercial success, including Left 4 Dead 1 and 2, Portal 1 and 2, and further iterations to the Half-Life franchise [6].

B. The Source Engine

In order to develop their games, Valve acquired the rights to use and modify the Quake game engine, published by id Software. The Quake engine was regarded as one of the premier video game engines of that time, powering the extremely popular and trendsetting First Person Shooter game Quake. This engine was the first to transfer from a two dimensional sprite based gaming system to a three dimensional world [5]. The borrowed game engine was heavily modified in order to better suit Valve's needs, and eventually became known as the Goldsrc engine. The following years at Valve were focused a combination of developing smaller titles as well as further enhancing the aging Goldsrc engine. After several iterations and releases, the Source engine was born from the outdated Goldsrc engine [7]. The Source engine has been used and is still being utilized on all Valve game releases since its inception. The modular nature of the Source engine lends itself to constant development and improvement. One of the most notable additions to the Source engine and all of Valve's published games is the integration of the Steam platform.

C. Steam

Prior to Steam, the release of an update or patch would result in the disconnection of a large portion of the users for some time as the game updated. Steam initially set out to better facilitate patch deployment and management. As Steam began to expand, it also gained more features and functions. Through time, Steam began to handle more than patch deployment, including digital distribution, multiplayer, digital rights management, community features, chat and voice functionality, and anti-cheat detection and resolution technologies. Eventually, the Steamworks API was also released, allowing developers to interface with the Steam platform. As Steam gained popularity, other game developers began to offer their game catalogs as downloads through it.

One of the largest draws of Steam is that it is both platform and machine independent. Since its inception, Steam has continually grown in both scope and user base. As of the beginning of 2012, Steam has 1523 games available through the store front [9], as well as 40 million active user accounts [10]. On January 2, 2012, Steam broke an all-time record by having 5 million concurrent players in game at the same time [11]. While Valve has never revealed any details about their market shares, a competing online distribution service Stardock estimated that Steam had 70% of the digital distribution market in 2009 [12]. Steam continues to see growth and development, and Valve has revealed no plans to abandon the popular service.

D. Hammer

One of the unique characteristics of the Source engine is its cooperation with the developer community. Many game engines choose to keep their tools and game mechanics from the general populous. They instead only allow contracted developers the opportunity to work with this proprietary software. This is not the case with the Valve Corporation. Most of Valve's tools, including Hammer, are often published with free access to anyone who uses their games.

The Hammer Editor is the official level creation tool used by Valve for all Source based games. It is free software available to any person who has purchased a Source based game. It is included as part of the Source Software Development Kit (SDK). Hammer is a replacement for the outdated Worldcraft tool which was used on Goldsrc games. Created by Ben Morris in 1996, Worldcraft's rights were acquired by Valve when they hired Morris a year later [13].

The Hammer editor allows a developer to create a map through the use of brushes, entities, and map properties [14]. Brushes are the most primitive of objects in a game level. They are primarily geometric solids such as blocks, rectangles, cones, and spikes. These brushes are the primary building blocks to all Source levels. Almost all large shapes and terrains are created through the manipulation of basic brushes. Small and more detailed objects are created through the use of models, a separate category of entity within Hammer. Entities are non-static, sometimes animate objects that are used for interaction as well as non-visible game data or logic needed to make a map come to life. There are two general types of entities: point and brush. Point entities exist logically at a point or points within the level. Examples of these entities include players, non-player characters, or lights. Brush entities are tied to a brush in order to exist, but modify its existence somehow. Some examples of brush entities include doors, elevators, ladders, or other moving interacting objects. Another example of brush entities are triggers, an invisible event that fires based on input from the player such as walking into an area or completing a task. By combining brushes and entities, a virtually limitless series of levels can be created.

Hammer also includes tools to compile raw map data into a format that is usable by the Source engine. By default, uncompiled maps are saved in the proprietary VMF format. This is a plaintext, human readable file format that stores information about the level [15]. In order to convert this text into information that the Source engine can use, several compilation steps are needed. There are four main programs which run to create a playable level: the game executable, VBSP, VVIS, and VRAD. The game executable parameter allows the user to specify which game and set of specific tools to use from the available Source based games [16]. For instance, Half Life 2 based games have different options and functionality than Left 4 Dead based games. Once the game parameter has been set, the map data is passed to VBSP. This tool converts a raw .vmf file into a compiled Binary Space Partition (BSP) file. This is the file type actually used by the engine to render the map. VBSP converts primitives such as brushes into polygons, generates visible sections of the map, creates props, and embeds entities [17]. Once this is completed, the .bsp file is passed to VVIS. VVIS embeds visibility data in the map. This is done by splitting the map into visleaves, which are small sections of the map that load one at a time, rather than all at once. This improves performance and load times significantly. VVIS also determines which visleaves can see each other for rendering order [18]. Once complete, the .bsp is passed to VRAD. The VRAD tool embeds lighting data into the map. Any user defined and dynamic lighting information is inserted at this point. The pre-compiled light is then propagated through the level through a radiosity algorithm [19]. Once all these tools are complete, the BSP file is ready to be loaded by the game engine and executed.

III. Hiding Data in Source Games

A. Why Video Games and Steam?

There are several criteria that make video game files excellent candidates for data hiding, the first of which is their size. When examining common files such as JPEGs, an uncommonly large file size can be an indicator of foul play, making it impossible to hide large file in these file types without raising the risk of discovery. The advantage of video game data is that large file sizes are common. A typical video game installation may contain more than ten gigabytes of data. With this large palette available, any number of files can be hidden within. Another factor which makes video game data a viable candidate for data hiding is its commonality. Video games, especially those with incredibly large sales volumes, are installed on millions of computers across the world. An investigator finding these files on a suspect machine may not immediately deem them as suspicious. Furthermore, because of the dynamic nature of video games, files on each user's machine are expected to deviate from the initial released version. This eliminates the possibility of an investigator using expected file contents or hash values to check for changes and therefore flag a file or system as potential evidence. Perhaps the biggest advantage video games have in data hiding is that they are entirely different on disk than when they are running. A simple text message in game could not be located in the code, yet obtained easily from an in-game window or overlay. Since most investigations are conducted on dead systems, these sorts of messages are almost untraceable. The only way for an investigator to mitigate this risk would be to load and execute every game level as part of an investigation for potential evidence. This sort of investigation is impractical and would never be used by an investigator in the field or in a digital forensic lab. All of these attributes make video game data an almost insurmountable challenge to investigators or unwanted data observation.

After establishing that video games provide an ample platform for data hiding, one questions remains: Why Steam? Of all the video game publishers in the industry, why target Valve's platform and Source engine? There are a number of factors that make Steam an excellent candidate for conducting this sort of activities. First is its widespread use. With over 40 million active users, the presence of Steam on a system should raise no suspicion for an investigator. An added benefit of this software is that it is cross-platform. The data embedded on files running on a Windows-based system will still be able to be transferred and utilized by malicious users on Mac or Linux platforms as well. The tools and methods for hiding the data are likewise cross-platform. Another contributing factor of utilizing the Steam platform is the tools for interfacing with and modifying data for the Source engine are also widely available and free to use. The Hammer world editor, for instance, comes free with any Source based game. This makes protecting data privacy more accessible for all users, even those with malicious intent. Because these tools have been available for some time, the file formats and behaviors are well understood and documented. Manipulating this data is easier, allowing for further tweaking and exploitation of game files and properties. One of the most unique properties of the Steam platform is the integrated social connectivity features. Steam handles much more than simply running video games, it connects users, distributes content, and has an integrated platform for sharing maps, mods, and tweaks. By utilizing this built-in functionality, malicious users can distribute game levels with hidden data on a grand scale. This activity would also not draw any unwanted attention from investigators, as content is commonly shared between users on the Steam network. By using the built-in tools, packaging, and sharing functionality, Steam is an all in one tool for hiding and transmitting data using video game files.

B. Steganography

In the realm of data hiding, there are two main approaches to data obfuscation. The first and most common approach is encryption. All encryption techniques follow the same basic strategy: data is sent through an encryption algorithm in order to generate ciphertext, then encrypted message is sent or stored for the recipient, and the ciphertext is finally decrypted with the key. The security of data depends entirely on the encryption algorithm and how well key is kept secure. An unintended recipient is able to see the ciphertext, but they cannot access the original information without the key. Steganography utilizes a different approach to data security. Rather than transforming data into an unreadable form, steganography hides data inside of a benign secondary piece of data. When data is hidden in this way, onlookers are unaware that there is any data hidden at all. Steganography is thus a form of security through obscurity. Other than the sender and receiver, nobody suspects that a message has been transmitted. Hiding data inside of Source game files is therefore a form of steganography.

C. Embedding Text with Brushes

Embedding data with brushes is the most straightforward method of obfuscation. To accomplish this, a malicious user can simply create a new brush and shape it in such a way as to form the words or letters that compose the hidden message. By adding several brushes, the message can be expanded from a single letter or word to a larger collection of text. While brushes are initially set as primitive shapes such as rectangles or cubes, more complicated solids can be created by utilizing Hammer's built in vertex and face edit tools, thus adding more tools to the malicious user's arsenal. Hiding text in Source games via brushes is a great tool for users who wish to obfuscate or share messages without being seen. The advantage of using this approach as opposed to standard encryption is that an investigator or onlooker will not be able to detect that there is data hidden in the level. Even if an investigator detects that data has been hidden in the game files, the data is untraceable on disk as it exists only in the geometry of the level. The main disadvantage of this approach is that it is tedious. It is also impractical with large amounts of information. For small messages, however, this approach is ideal.

D. Embedding Text with Overlays

Game overlays are messages and dialogs which appear in-game as the player navigates through the game map. They are typically used to provide a player with important information, hints, or instructions. A malicious user may manipulate this functionality to embed messages which have no bearing on the game, but are instead intended as hidden data. In order to embed and hide in game text overlays, two entities are utilized at a fixed point in the map: *env_instructor_hint* and *info_target.*

Env_instructor_hint is a point entity in the Left 4 Dead version of the Source engine [20]. This entity exists to provide in-game hints for players via a popup on screen. This popup is primarily text based, although a small image may be inserted. By using this entity and creating custom text, any message can be embedded in the game. Any number of these entities can be added to a given level, creating a virtually limitless space for text to be hidden.

After inserting an *env_instructor_hint* into the map, the entity can be customized through several variable attributes in Hammer. The most useful of these variables is Caption. Caption holds the text that is ultimately displayed on screen and is where a malicious individual can embed hidden messages. This text can be combined with one of many predefined images that are included in the Onscreen Icon variable. There are many other variables within env_instructor_hint which define the type of text, size, color, and pulsation, and other attributes. With this entity alone, a hidden message can be easily embedded in a map which is as simple as a cube with the player inside.

Although *env_instructor_hint* is the mechanism used for hiding messages, it does not exist on its own. The env*_instructor_hint* entity does not yet have a physical place in the map to display. *Info_target* serves this purpose. *Info_target* is the physical placeholder to which *env_instructor_hint* is bound [21]. Wherever the *info_target* entity is placed is where the user will see the text defined in *env_instructor_hint* displayed on screen. Combining these two entities can be useful for hiding data in a specific location in a map that the intended user knows to look for. By utilizing env*_instructor_hint* and *info_target*, malicious users can transform a custom game map into a container filled with malicious text messages.

E. Embedding Images with Textures

Child pornography is one of the most grievous cybercrimes facing law enforcement and digital forensic investigators. These investigators analyze systems for this type of content to expose the offenders. Malicious users employ encryption, steganography, and other means of obfuscation to thwart investigator's attempts to uncover the proper evidence to convict these criminals in the court of law. Hammer can also be used as a tool to hide potentially illegal or otherwise malicious images.

While embedding text into a Source map file is fairly straightforward, adding images requires more involved work. Any image applied to a surface or brush in a map is referred to as a texture. To begin examining how these files are hidden, it is necessary to understand how the Source engine handles images. Most images are saved as predefined file types that are familiar to most computer users. Examples of these include JPEG, PNG, and GIF. The Source engine does not interface with these types of images directly. Instead, it uses a proprietary format: VTF.

Valve Texture File (VTF) files are stored in a format different than more widely used image formats in several key areas. The most notable characteristic of the VTF format is that the total size of the image must be a power of two as measured in pixels. For example, images must be 2x2, 4x4, 8x8, 16x16, and so on. The dependency on square images can be a hindrance to users embedding data; however most images can be made square through selective cropping or the addition of white space. The reason for the reliance on square images is the presence of mipmap data in the VTF file. Mipmaps are smaller, lower resolution versions of the original image also stored in the file. These mipmaps are used by the Source engine to render the image differently at varying distances in the game. The lower resolution images are used at the farthest distance, with increasing resolution as the image comes closer on screen. This improves rendering speed and processing performance of the game. Each respective mipmap is half of the previous mipmap, creating the reliance on a power of two size constraint. VTF files also contain other additional information used by the Source engine, including a bump map scale, a low resolution copy of the VTF for color rendering, and a reflectivity value used by the VRAD program in determining final rendering appearance [22]. More complex textures such as environment maps and volumetric textures include even more data, all of which is stored in the VTF file.

Hammer does not include a tool for creating custom textures that are not already bundled with the engine. Luckily third party tools exist to convert common image file types to VTF files. One of the most prevalent of these is VTFEdit. VTFEdit is a free, open source tool which can create a VTF file from almost any popular image file type. It can also edit flags and attributes within a VTF file. VTFEdit utilizes the open source VTFLib library. VTFEdit not only can convert to and from the VTF file format, but has additional functionality which allows it to edit VMT files discussed in detail below. VTFEdit has the ability to handle files in batches as well as create all mipmaps and VTF file headers utilized by the Source engine [23].

The Source engine does not only use VTF files for rendering textures in-game. Each texture file is also paired with a corresponding Valve Material Type (VMT) file. These files are text based files which serve as a set of metadata for the texture they correspond to. This allows the Source engine to determine how to render the textures properly. Attributes included in VMT files include texture names, physical surface types, shader parameters, fallbacks, and proxies [24]. Although these files are usually fairly simple, they can be used to hide either plain text or encrypted text. While this type of text would be easily viewable on disk to an investigator, it can still serve as an additional container for malicious data.

Once a texture has been generated from VTFEdit, it can then be embedded into the map within Hammer. This is a simple process done through Hammer's map creation interface. The texture application can be invoked to navigate through the engine's built in textures. Depending on where the custom texture is stored, the creator can navigate to the directory and select the custom texture. Applying the texture is as simple as selecting the appropriate brush or entity and clicking the Apply Current Texture button. Once applied, many of the texture's properties can be edited from within Hammer. Once compiled, the texture and any hidden information it or its corresponding VMT file contains is permanently embedded in the level.

The file extension used by an image on disk is of little importance to an investigator. The investigator's primary objective is to identify the file format based on its content rather than file extension. This is done commonly though the use of file signatures. JPEG images, for instance, have a common header and footer in the code of every image. This header and footer do not change due to a change in the file extension. Investigators can utilize this fact by searching for the file header rather than the file type on a suspect system. In the case of JPEG, the file signature is 'ÿØÿà', or FF D8 FF E0 in hexadecimal. When converting an image to a VTF file, the file signature and header previously used by the image is lost. This is true for not only JPEGs, but all other file types. By eliminating these file signatures, an investigator's most common and effective method of uncovering images is proven ineffective. This makes the recovery of illegal or malicious files hidden within a Source game level a greater challenge. The reliance on outside tools and the complexity of the process required to embed custom images makes this method less than ideal for a malicious user. The time required to perform this type of steganography is great.

F. *Distributing Maps*

After a map and all of the corresponding assets are created, packaging and distribution can take place. Valve has a proprietary file format for this process, called Valve Pack (VPK). This file format is relatively new replacing the outdated GFC format formerly used in Source games [25]. VPK files are packages which contain all of the necessary components for custom maps to be installed and run. This includes the map's BSP files, navigation logic files, textures, and a few identifying pieces of metadata used by the in-game UI. It is important to note that VPK data is archived, and therefore typically tightly packed. This format allows developers to share their data through a single file download which makes download and installation of game content seamless and easy for the end user. In order to create a VPK file, Valve has released a free tool as part of the Source engine SDK. This tool, simply called VPK, uses a directory containing the game files to be packaged as input and outputs the completed package. It can be run as a command line tool, giving it the ability to be used in batch programs to output large amount of game packages. It can also be used to list and modify the contents of a VPK file [26]. Game users can either double-click the VPK package to install it into the game via the operating system or manually add it by placing it in the 'addons' subdirectory of the game's installation files. For an investigator, this file provides a one stop shop for potentially malicious game content.

IV. INVESTIGATING SOURCE GAMES

A. *The Forensic Process and Tools*

As demonstrated throughout this paper, Steam maps can be used to store and transmit sensitive and potentially dangerous and illegal data. From an investigative standpoint, these files hold a potential treasure trove of information and evidence. As an investigator, extracting and examining this evidence is crucial. There are many tools which can be used accomplish this task. One of the most popular and widely used of them is the Forensic Tool Kit (FTK).

FTK is a piece of enterprise level digital forensics software used by many investigators in the field and in crime labs across the U.S. This software package has been forensically tested and is verified as sound for use in investigations. In order to assume the role of an investigator, Steam game files will be examined with FTK. The files used for this process were extracted from a hard drive and archived, which is typical of what an investigator will do with a suspect machine, rather than working with a live system.

One of the main problems for investigators of Steam game files is the sheer amount of data to be processed. Every texture and map file potentially holds condemning information. Deciding which of these files hold evidence is the first obstacle to overcome as every Source based game comes pre-packaged with hundreds of textures and many map files. It is therefore the obligation of an investigator to determine which of these files hold malicious data and which are benign.

The first order of business when investigating Source game files is to determine how they are stored. As stated above, game files may be stored as a single archived VPK file or as a series of directories in the installation location of the game. To locate VPK files on disk, an investigator can search for all files with the VPK extension. In the case of a suspect changing the file extension to cover up data, the investigator can also find these files by the VPK signature, 0x55aa1234. Once the

VPK file has been found, tools can be used to unpack it to its constituent file structure. The unpacked file structure is much easier to work with, allowing an investigator to search in the appropriate folders for the files they need. One of the tools available to unpack these files is GCFScape.

GCFScape is a free utility that is part of the Nem's Tools pack of Source map editing and enhancement software [27]. It is open source and can be used freely by anyone. This tool allows users to view, modify, and extract the underlying file structure from VPK files. For an investigator, it can be used for its extraction functionality. Although it is not a fully verified forensic tool, its open source nature could provide for verification in the future should an investigator need to pursue this avenue. After GFCScape has been used to extract the custom map files from a VPK file, they can then be examined by the investigator. Unfortunately, FTK cannot natively display the visual contents of any of the proprietary Valve formats, so hexadecimal searching is the most efficient way to sort through these files.

B. Investigating Data Hidden With Brushes

It has been shown that data can be hidden in a level by manipulating brushes and geometry to create words. Unfortunately, this type of hidden data can hardly be uncovered by an investigator. The level geometry is stored in integer format based on its vertices and their location in the map. This information is useless on disk to an investigator. The only way to view and use this data is to load the map into the game and run it. This method of an investigation is often not feasible or practical for an investigator; however it may be used as a last resort.

C. Investigating Data Hidden with Overlays

An unpacked VPK file contains many resources, including textures, maps, models, and metadata. Each of these files may contain evidence hidden by one or many of the above methods. The easiest of these to recover is embedded messages hidden via in game pop ups. These messages are stored in the *mapname.bsp* file. By examining the hexadecimal contents of this file, important pieces of information can be uncovered.

Within a BSP file, entities are defined in entity lumps. Each of these lumps contains defining information about an entity, including its location and properties. This information is stored in plaintext as it appears in game. By using the identifying fields in the entity lump, an investigator can recover hidden messages. Messages hidden with the *env_instructor_hint* entity as discussed previously can be found by searching for the keyword *hint_caption*. The *hint_caption* field contains the actual text displayed on screen in game as a popup hint. In the BSP, an example of a recovered *hint_caption* is:

```
"hint_caption" "Any message may be hidden
in game as text!"
```

By recovering all instances of this keyword, an investigator can uncover hidden messages from a Source map hidden using *env_instructor_hint*.

D. Investigating Data Hidden with Textures

Investigative parsing of map data for hidden images can be accomplished in a similar manner to searching for text data. By utilizing the file structure and container formats, an investigator can identify files and then extract them for investigation. Because images included in game files do not have common file signatures, they will not be properly flagged in most investigative software. This limitation creates the necessity for an investigator to manually find these files using the file signatures.

To uncover hidden images in Source game files, the investigator needs to first identify custom textures embedded in the map. This can be done by first expanding the VPK file with GCFScape as discussed above. Once expanded, all custom textures can be located in the *materials* directory. By utilizing the underlying file structure to identify custom textures, the investigator can effectively narrow down the search for suspicions textures from every texture in the game to a small fraction of that number. Alternatively, the investigator can find all VTF texture files by searching for the hexadecimal VTF file signature "VTF\0".

With the suspect texture files identified, they can then be inspected by the investigator. Again, FTK does not support natively viewing these files. An outside program such as VTFEdit would need to be used to open this file. Any malicious images can be viewed and exported from this program. While VTFEdit is not forensically verified, it can still be utilized in an investigator as no digital forensic suites can currently parse this data natively.

V. CONCLUSION AND FUTURE WORK

The field of digital forensics is a double bladed sword. On one side there is the concept of privacy, as individuals believe that they have the right to protect their own data however they see fit. This can include encryption, steganography, and other means of obfuscation. On the other side, there are investigators and law enforcement attempting to prevent and uncover individuals using these means to conduct illegal or malicious activity.

In the name of individual privacy, many tools and software have been developed in order to hide or otherwise prevent viewing or tampering by anyone other than those intended. Many file types have known exploitations or loopholes which allow for data to be manipulated in order to protect privacy. Although this is a pain for investigators, it is a necessary evil. One vessel for data hiding that has not been developed or researched in depth is video game data.

This paper demonstrates both ends of the spectrum of data privacy and security. It shows how a concern citizen may use game files to store information that they want to secure from prying eyes. While it is true that this privilege may be abused in a malicious way, this fact is not guaranteed. This paper also demonstrates how an investigator may conduct an investigation in the face of these new techniques for data hiding in video game files. It further demonstrates the need for investigative technologies to continue growth in order to keep up with the multitude of computer crimes being committed constantly across the nation and the world.

Computer crime is an ever-growing, ever changing venture. Just as law enforcement has caught up to the current criminal technology, new technology is introduced. This ongoing game of cat and mouse necessitates the constant development of tools and methodologies for fighting computer crime. This concept can be demonstrated with the utilization of Steam and Source based games to obfuscate data as well as the need for tools to address the issue of hidden data.

Three main methodologies for hiding data in source games are demonstrated in this paper; however there are potentially countless more methods available. For instance, steganography in the raw data files of Source maps can be further investigated and implemented as well as data in transmission during online or LAN games. This paper also focuses on one game engine only. There are many more game platforms and engines such as Origin, Stardock, and Gamefly which may have hidden data which also need further research.

On the investigative front, there is also room for future research and development. As mentioned above, neither FTK nor any other investigative tools can natively display or manipulate Source game files. Tools to analyze data in transmission from these games are also necessary to intercept and analyze potentially illegal data in transfer. The addition of this functionality to these tools will greatly improve their ability to investigate and process game files for hidden data. Forensically testing and verifying these programs will create tools which can be used in court to convict criminals or exonerate innocents.

REFERENCES

[1] M. Fossi and T. Mack, "Symantec Internet Security Threat Report: Trends for 2010," Symantec Corp., Moantain View, CA, Tech. Rep. 21 182883, Apr. 2011

[2] Entertainment Software Association, (2011). Essential Facts about the Computer And Video Game Industry [Online]. Available: http://www.theesa.com/facts/pdfs/ESA_EF_2011.pdf.

[3] Entertainment Software Association, (2011). Industry Facts: Economic Data [Online]. Available: http://www.theesa.com/facts/econdata.asp.

[4] Valve Corporation, (2010). Welcome to Valve [Online]. Available: http://www.valvesoftware.com/company/index.html.

[5] T. Bayer, (2010). 14 years of Quake Engine: The Famous Games with id Technology [Online]. Available: http://www.pcgameshardware.com/aid,687947/14-years-of-Quake-Engine-The-famous-games-with-id-Technology/News/

[6] M. Thomsen, (2009). Ode to Source: A History of Valve's Tireless Game Engine [Online]. Available: http://pc.ign.com/articles/102/1027317p1.html.

[7] A. Capriole and J. Phillips, (2008). The History of Valve [Online]. Available:http://planethalflife.gamespy.com/View.php?view=Articles.Detail&id=121.

[8] Warf!y, (2011). About the Steamless CS Project [Online]. Available: http://v5.steamlessproject.nl/index.php?page=about.

[9] Valve Corporation, (2010). Games [Online]. Available: http://store.steampowered.com/search/#category1=998&advanced=0&sort_order=ASC&page=1.

[10] K. Mudgal, (2012). Valve Releases PR; Steam Userbase Doubles in 2011, Big Picture Mode Coming Soon [Online]. Available: http://gamingbolt.com/valve-releases-pr-steam-userbase-doubles-in-2011-big-picture-mode-coming-soon.

[11] T. Senior, (2012). Steam Hits Five Million Concurrent Players [Online]. Available:http://www.pcgamer.com/2012/01/03/steam-hits-five-million-concurrent-players/.

[12] K. Graft, (2009). Stardock Reveals Impulse, Steam Market Share Estimates [Online].Available:http://www.gamasutra.com/php-bin/news_index. php?story=26158.

[13] Hammer Editor Version History (2010) [Online]. Available:https://developer.valvesoftware.com/wiki/Hammer_Editor_version_history.

[14] Mapping Overview (2010) [Online]. Available: https://developer.valvesoftware.com/wiki/Introduction_to_Editing.

[15] VMF Documentation (2012) [Online]. Available: https://developer.valvesoftware.com/wiki/VMF_documentation.

[16] Hammer Game Configurations (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/Game_Configurations.

[17] VBSP (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/Vbsp.

[18] VVIS (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/Vvis.

[19] VRAD (2012) [Online]. Available: https://developer.valvesoftware.com/wiki/Vrad.

[20] Env_Instructor_Hint (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/Env_instructor_hint.

[21] Info_target (2012) [Online]. Available: https://developer.valvesoftware.com/wiki/Info_target.

[22] Valve Texture Format (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/Valve_Texture_Format.

[23] VTFEdit (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/VTFEdit.

[24] Material (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/Material.

[25] VPK File Format (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/VPK_File_Format.

[26] VPK (2011) [Online]. Available: https://developer.valvesoftware.com/wiki/VPK.

[27] R. Gregg, (2006). AboutGCFScape [Online]. Available: http://nemesis.thewavelength.net/index.php?p=25.