SmartFoxServer 2X
Security White Paper

March 2013
(last revision March 2016)

Author

Marco Lapi
The gotoAndPlay() Team

Introduction

In this document we will take an in-depth look at the many security concerns that an online multiplayer game may encounter from both a client and server side perspective.

Hacking attempts can take many different forms, from rudimentary packet flooding, to malicious requests up to full-scale denial of service (DoS) attacks.

SmartFoxServer 2X offers a rich set of security tools out of the box, that will help preventing and mitigating all kinds of attacks and build a robust online experience for all users.

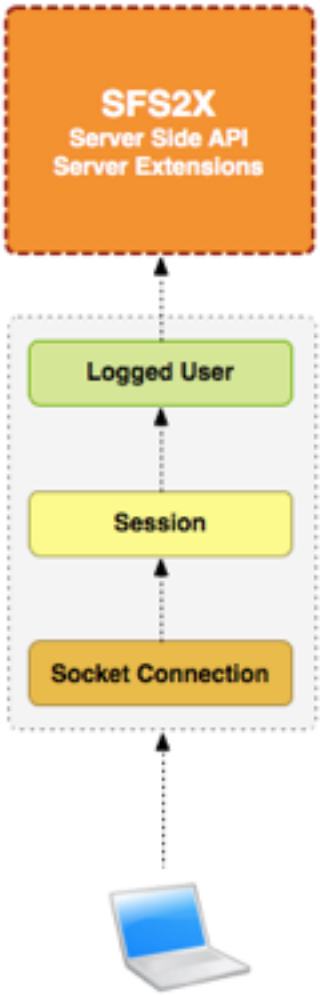
Additionally this paper will provide best practices for securing SmartFoxServer from malicious users, harden the custom server side code and ultimately protect the system against massive attacks.

Starting from the wire

We will start our journey through the many security concerns at the lowest level possible, following the process that allows a simple socket connection to create a Session, then a regular User and finally interacting with the server and other clients.

At each step we will discuss the types of attacks that can be performed and how we can detect and counteract them effectively.

We will begin by reviewing the basic steps that a user goes through via the SmartFoxServer Client API in order to interact with the server and the other players in the system.



Socket Connection
represent the simple act of opening a TCP socket towards the server. Any client is authorized to accomplish this step because the server is listening on a public IP address.

Session
a handshake is exchanged between client and server, the client must be able to speak the SFS2X protocol and provide a valid API version.

Logged User
the client has provided valid credentials and has joined a Zone in the server which represents one of the applications running in the system. At this point the User can interact with other clients.

The Socket Connection Level

You have have deployed and configured SmartFoxServer 2X on the production server. SFS2X is running on a public IP address, listening for any clients to connect and interact with the system. At this stage we have no idea about who is connecting and what their intentions are.

In our previous diagram we have assigned an orange color to this stage because it is actually one of the most vulnerable from a security point of view.

Since every each TCP connection has a “cost” in terms of physical server resources (mostly RAM and some CPU) we want to make sure that only legitimate clients will be able to use these resources.

In order to understand how to secure the Connection Level let’s take a look at how malicious users can take advantage of a “free connection” to cause damage in our system.

» Resource Starvation

This is the most common attack that can be performed at this level. It consists of launching thousands of sockets towards the server machine in order to allocate as many resources as possible in an attempt to ultimately bring the system to a halt.

This scenario is quite frequent and can be conducted with a couple of spare machines by any “script kiddie” as well as with a massive amount of dormant “zombies” via a malware or virus that has spread his infection. In the latter case the malware will first silently replicate itself, over a period of time, on as many vulnerable clients as possible, then wake up in a sycnhronous fashion and perform a massive attack towards one or more targets.

In any case these attacks can be terribly effective and the history of the internet abounds with anecdotes of web services being crashed at will by anonymous hackers.

There are at least three lines of defense to minimize the risks of resource starvation:

1. **SmartFoxServer 2X configuration**
   - 1.1. **IP Filtering:** denies multiple connections from the same IP address. The number of clients originating from the same IP can be fine tuned to avoid massive attacks from a few machines. This alone can be very effective.
   - 1.2. **Black lists:** if you have detected bad behavior or anomalous traffic from a specific set of IP addresses, you can add them to the SmartFoxServer’s IP black list to deny them access at the socket level.
   - 1.3. **White lists:** you can also specify IP addresses that instead are fully authorized to open as many connections as they wish towards the server. This way you can, for example, perform stress test towards your live server without the risk of opening security holes.

2. **Traffic monitoring**
   2.1. **Hosting bandwidth monitor:** the hosting network monitor is the first place to look for anomalous traffic, unexpected peaks etc...

   2.2. **SmartFoxServer Logs:** the server side logs are also useful a resource for discovering malicous users. You can monitor the logs directly via the AdminTool to check for warnings related to possible hacking attempts.

   2.3. Our **Analytics module** can also show detailed statistics about user connections, session duration, geolocation and highlight possible anomalous traffic.

3. **Traffic shaping**

Traffic shaping is an advanced technique for managing network traffic that allows to optimize bandwidth resources, improve latency and protect from DoS attacks.

Also known as “packet shaping” this technique allows to establish many types of socket-level limitations for network communications to avoid congestions and bandwidth waste. The service can be configured to recognize traffic by protocol and make sure that clients don’t exceed specific rates or transfer limits.

The whole subject of network management and traffic shaping could take several more papers and it is highly technical in nature. Normally a multiplayer developer doesn’t really need to learn or know how to implement it. The best idea is to work with the hosting provider to find a solution that suits our application requirements and type of traffic.

» Protocol security

Another important topic we are often asked about is the security at the “wire-level” and how hackers could spy the traffic, decode the protocol and inject malicious packets or find ways to send forged requests.

In order to address the complexity of these attacks and their likelihood there are several important points that need to be clarified:

- **Persistent connections**
  For starters clients connection are persistent and socket-based, therefore the possibility for an external attacker to spy such connection is vanishingly low. In order to be able to watch someone’s network traffic the attacker should have already gained significant privileges within the local network and/or the victim’s machine.

- **Protocol encryption**
  Since SmartFoxServer 2X 2.10 the binary protocol provides full cryptography support via TLS 1.2, the current industry standard for secure internet communications.
  This works very similarly to HTTPS, where a shared secret is initiated between client and server via a secure public-key exchange (**Diffie-Hellman**). The two parties then employ the shared secret to encrypt the communication using a symmetric cryptographic algorithm such as AES.

- **Packet Injection / Man-in-the middle attacks**
  These types of attacks consist in redirecting the client's traffic to a malicious host which sits in the middle of the communication and watches all messages. The "man in the middle" can also selectively edit parts of the exchanges on the fly and inject new data, unbeknownst to both client and server.

  Activating the protocol cryptography is the only way to defeat these attempts 100%, since SSL certificates render the man-in-the-middle attack useless.

  It's also worth noting that these attacks are quite difficult to pull off unless the victim's network is already compromised. If the attacker works from within the same local network of the player (by way of an authorized access or by breaching it) there are higher chances that this attack may succeed, if the secure protocol is not activated.

  For more details on how protocol encryption and SSL certificates work we recommend consulting the **specific section of our documentation**.

- **Conclusions**
  Generally speaking protocol-level attacks are not very common as they are quite elaborate to pull off and require multiple pre-existing vulnerabilities in the target's network and/or computer.

  The best practice to avoid any protocol level hacking is to always activate protocol cryptography for all active Zones in the server.

  This however can come at a performance cost, depending on the type of hardware in use and the volume of traffic handled by the server. For instance, a fast action game with dozens of positional updates per second (per user) is best run without encryption, to avoid incurring in high performance costs. Also positional updates are easily validated via server side code and cryptography wouldn't add any special advantage.

  On the other hand for applications that involve gambling, trading or otherwise exchanging some sort of currency while playing, it is highly recommended to engage protocol cryptography to avoid any malicious attacks at the protocol level.

The Session Level

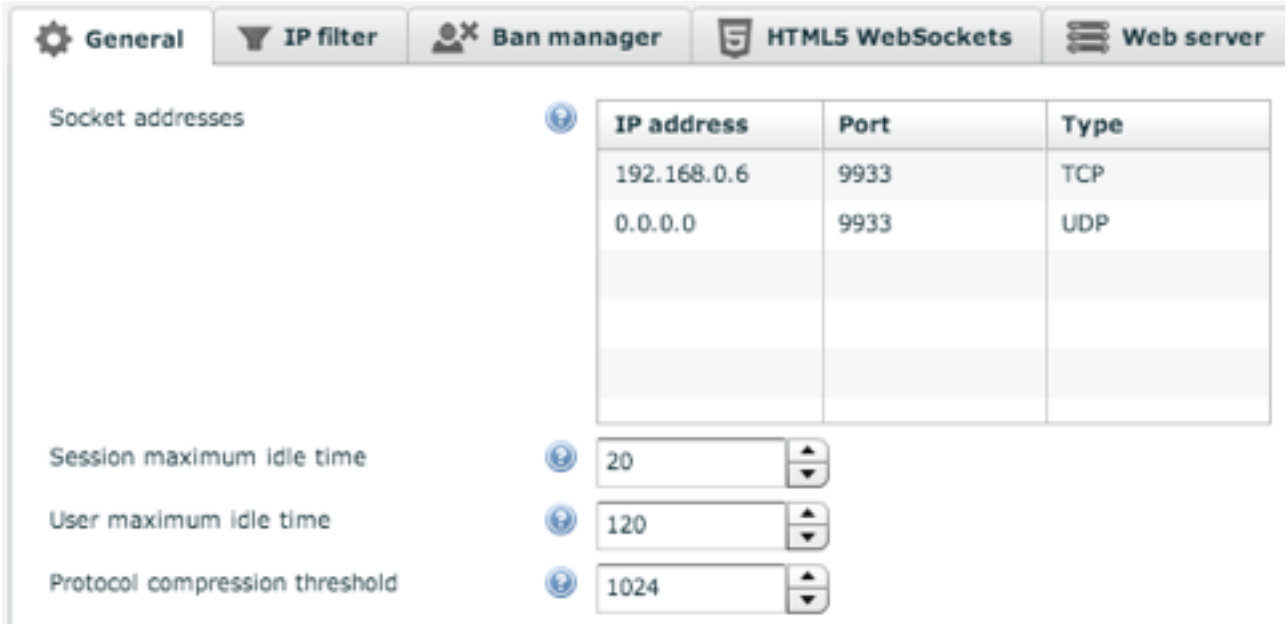
A Session is required to start the communication with SmartFoxServer 2X. Similarly to a web server it provides a unique identifier for each client connected in the system and, being socket-based, it is intrinsically more secure than HTTP sessions which are based on cookies.

The session is initiated from the client side by sending an handshake request which allows the server to validate the client type and version and adding an entry in the Session Manager.

Once the basic socket connection is promoted to Session the client is allowed to send one new type of message: the **LoginRequest**. Any other command at this stage is refused. At this point the client is supposed to login into a Zone relatively quickly.

By default the user is expected to login within 20 seconds, after which a timeout will kick in shutting down the socket and removing the Session. This way SmartFoxServer is able to quickly get rid of all clients that are wasting time using socket and Session resources without being active.

This timeout value is configued under the Server Configurator inside the AdminTool and is called *Session maximum idle time*



In general we don't recommend to change this value unless there are specific reasons.

We can now review some of the most often asked questions as regards the security of the login phase. We will take in consideration the most common scenario of a database-driven login system handled by a custom Extension.

» SQL Injection and database attacks

SQL Injection is a common type of attack for every system that exposes a public login interface, SmartFoxServer included. Depending on how the login server code and UI work, it might be vulnerable to specifc user input aimed at inserting malicious code in the server side SQL code.

The SmartFoxServer 2X API provide specific methods for running parameter-based DB queries that automatically sanitize the user input, defeating any code injection attempt.

We are referring to the SFSDBManager class, exposed in each Zone object (and accessible via Extension) which offers two methods:

```
public ISFSArray executeQuery(String sql, Object[] params) throws SQLException
public void executeUpdate(String sql, Object[] params) throws SQLException
```

For developers who need to work directly at the connection level we recommend to always use prepared statements via the relative PreparedStatement class from the JDBC library.

You can check the SFSDBManager class documentation for more details.

» Password security

Another common concern is that the user's credentials might be captured by someone peeking at the network traffic.

This is however a highly unplausible scenario for at least two reasons:

- **Encryption**: passwords are not sent in clear. They are encrypted with a unique session token and hashed, which is an one-way only encoding process. If someone is watching the network traffic he won't be able to decode anything.
- **Socket connections can't be spyed upon easily:** being able to watch someone else's socket traffic isn't small potato. Unless the target's system is already badly compromised (e.g. network intrusion, privileged system access, etc...) it's not possible to spy someone else's communications.

» Session Spoofing

This type of attack has become popular in the internet world thanks to the ability of hackers to hijack the session cookies transmitted by an HTTP server.

SmartFoxServer, however, doesn't use cookies for session management. Instead each user is identified by his physical socket connection, with its unique IP address and port.

In other words, this type of attack is simply not possible with SmartFoxServer.

» Multiple logins

Another common trick to obtain better scores and achievements in turn based games is to login with multiple clients operated by the same user. This is usually done by registering multiple accounts and then playing against oneself to gain more experience points, skills or virtual money depending on the type of game.

Counteracting these kind of stratagems can be more tricky and it usually shifts the responsibilty on the developer's code and the game logic.

One efficient way to reduce this way of cheating is to set the server's IP address to 1, thus making it possible for clients to connect one at a time from a specific address. This approach, however, has adverse consequences that should be considered, namely the fact that people connected behind the same router will not be able to play together. The scenaio is quite common for many small and corporate officese, schools, institutions etc... Therefore enforcing a 1-user-per-IP rule might be counterproductive.

There are also more sophisticated techniques to avoid multiple clients from the same users, such as requiring a valid cell phone number at registration time that will be used to send a security token via text message. The token can then be requested to activate the account in the system.

The User Level

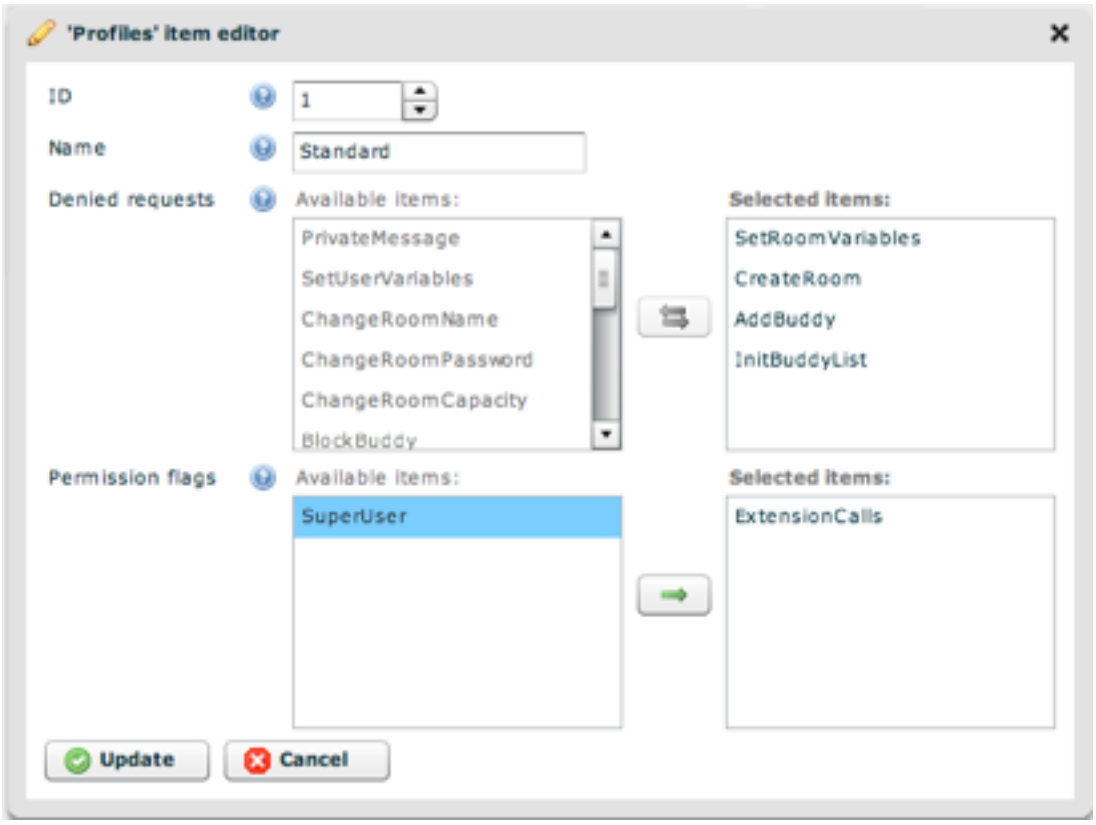
Once a client has passed the login stage he is finally able to send all types of requests to the server, interact with other players and communicate with custom Extension(s) available in the Zone and related Rooms.

In order to provide a more fine-grained security system, SmartFoxServer 2X provides a Privilege Manager that can be customized per Zone to limit the interaction with the Server. Each permission profile can configure a list of denied API calls for each User level. For instance, we could prohibit the creation of Rooms and Room Variables for guest users, and allow them only for registered users. Moderator and Administrator messages could be denied for everyone except those two privileged categories etc...

In other words the Privilege Manager allows the administrator to create any number of User profiles and decide, for each of them, which requests are allowed and which aren't. These profiles can then be managed by the login code and applied to each User based on any custom parameters (e.g. a privilege ID coming from the DB user profiles)

Similarly the ability to recognize the privilege level of each user allows the cusom server side code to limit or deny access to specific calls.

Privilege settings are easily configured via the **AdminTool** > **Zone Configurator**



All the details about the Privilege Manager are found in our online documentation.

» AdminTool security

Since the SmartFoxServer administration panel is the core of all server settings and monitoring, it is best to restrict its access as much as possible and avoid exposing it to the world.

The best practices to harden the AdminTool access are:

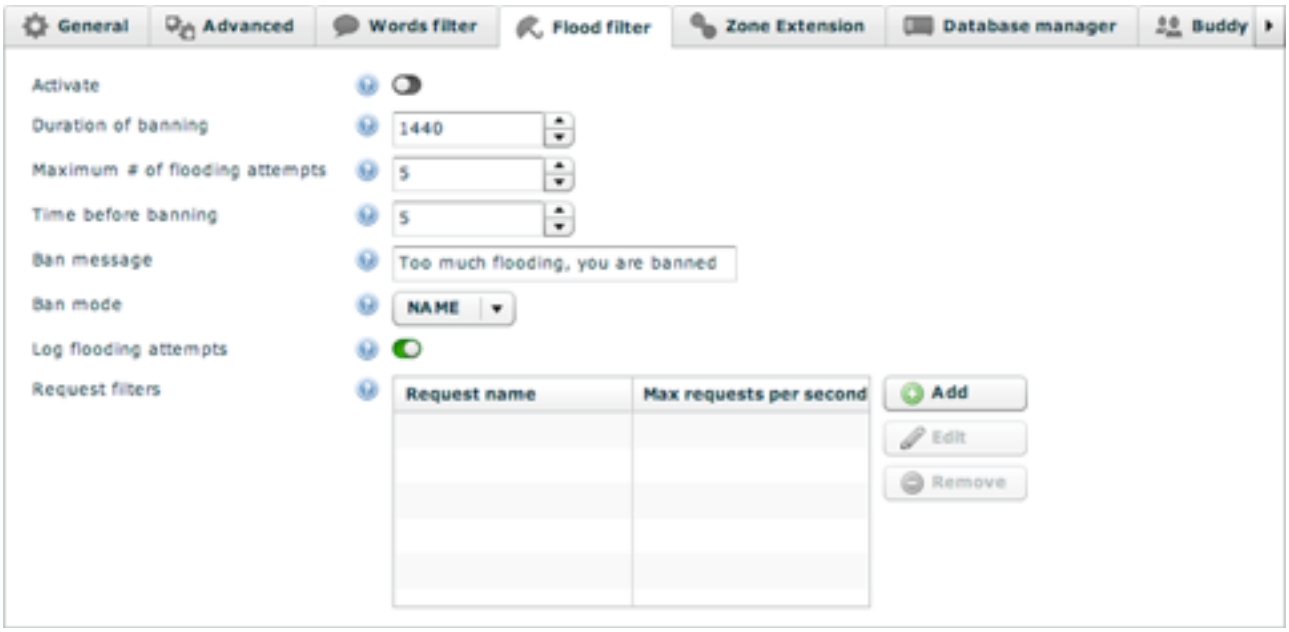
- Choosing a different TCP port exclusively for the tool.
- Providing a list of authorized IP addresses that can connect as Admins.
- Generating strong admin passwords (8-12 alphanumeric and numeric random chars).
- Routinely reset and regenerate the passwords (e.g. every 4-8 months).

» Anti Flood filters

We have already encountered the problem of clients trying to spam the server with connections or packets, in the first stage of our journey. Now that the client is logged in the system and free to send requests, we need to make sure that no one can flood the server with useless requests.

The flood filters in SmartFoxServer 2X allows the administrator to setup a threshold (expressed in requests/sec) for each request available in the system, and set rules to warn the User and ultimately kick him out of the system.

After a number of warnings the user can finally be banned.



You can learn how to configure the flood filter from our documentation

» Bad Words Filtering

SmartFoxServer 2X can auto-moderate bad language and filter swear words from public and private chat messages, room and user names, buddy messages and more.

Via a centralized Words Filter the administrator can provide a customizable list of regular expressions that can be selectively applied to different types of messages exchanged between users. Similar to the flood filter, warning messages can be automatically sent to the offending users, which in turn will trigger either a *kick* or *ban* from the system.

The Words Filter can work in black-list mode (i.e. catching the unwanted expressions) or as white-list mode, allowing to provide a dictionary of permitted words and expressions. Anything falling outside the provided list will automatically be filtered. This is probably the best strategy for web sites oriented towards kids and teen agers.

Since battling swear words is a difficult task in itself, SmartFoxServer 2X provides other strategies to improve the end result:

- **Chat History and Monitoring:** the server side code can listen to any chat event and track each message for further inspection or for keeping chat histories.

- **Custom filtering:** via the SystemController Filters, developers can add custom code in front of any request coming into the server. This way they can plug additional filtering logic to chat messages and/or use third-party solutions even running as remote web-services.

You can learn more about these word filtering techniques from our documentation:

- Word Filter configuration in the AdminTool
- Server side events
- SystemController filters

» Rooms and Variables Flooding

In order to prevent malicious use of Rooms and server Variables (i.e. User/Room/Buddy variables) the ZoneConfigurator in the AdminTool provides limits on the dynamic instantiation each of these object on a per-user basis.

By default the number of Rooms and Variables created by a single client is already set to a conservative value. We recommend to adjust these settings in accord to the requirements of your application.

» Request Size Limits

In order to avoid nasty large-packets being sent to the server with the purpose of slowing it down, the default SmartFoxServer 2X configuration sets the limit of an incoming packet to 4096 bytes (4KB), which includes compression. You can adjust this setting according to the requirements of your games, keeping in mind that very permissive values might open the doors to large-packets attacks. Recommended values are anything below 50-100KB.

» Security at the Extension Level

Since Extensions contain the core logic of our games it is of the highest importance to apply best coding practices to avoid attacks and cheating:

- **Input validation:** this alone represents more than 50% of the security in our Extension code. Making absolutley sure that each parameter coming from the client is valid and within range can save us a lot of headaches. Clients can be hacked, modified or even rewritten from scratch with the sole purpose of taking advantage of server side vulnerabilities.

  We usually suggest to create a **validate(...)** method for each request in your handlers where you can sanitize and check each parameter and throw detailed exceptions in case anything is out of order. By logging the username, IP address etc... you will be able to investigate who is trying to hack into the system.

- **Flooding detection:** similarly to what we have discussed in the previous section, the problem with flooding also applies to Extensions. Since SmartFoxServer cannot predict the type of traffic generated by each Extension request, developers will need to take of flooding attacks via customized code.

  This can be easily done by storing the time of the last request in a private server-side User Variable, and then taking appropriate action based on the application logic.

  Kicking and banning functionalities are available from the sever side API

- **Storage corruption:** if the server side code supports custom storage of any kind (e.g. high score tables, user profiles, user uploads) we will also need to make sure that all the data coming from the client conforms to the types and sizes we have chosen for the storage support (e.g. a database).

  For example custom avatar images should comply with the limits in size and file types, custom string values should not exceed the expected lenght and shouldn't contain characters that might affect or break the UI (e.g. HTML text containing malicious javascript).

Securing the Client

We have already mentioned that the client side is the weak point of the communication and discussed how attackers can either modify an application or write a custom one that mimicks the original in an attempt to exploit potential weaknesses.

In this section we'll bring our attention to a number of strategies to strengthen the client application and improve the overall security of the system. We'll start with what we feel is the most secure type of clients and proceed with the most vulnerable down the line.

» Web based clients

We consider this family of applications slightly more secure than the others in this list for a number of reasons. In particular we are referring to technologies such as Flash and Unity3D which offer great possibilities for multiplayer games.

The reason why we feel these clients are slighlty more secure is because they are served as compiled binaries from the developer's HTTP servers and use cross-domain policy files to control access to external resources.

The ability to update clients as often as it is necessary and the restrictions imposed by the cross-domain policies allow to avoid the most common attacks and to quickly react to bugs and vulnerabilities with new patches.

In order to further strengthen web clients there are a number of strategies that can be applied:

- **Code obfuscation**: will render decompilation much harder and the re-compilation of modified clients almost impossible.
- **Strict version control**: the server can verify the client release and deny access to any client with a wrong or unknown version number.
- **Socket based transfers**: the client can silently transfer small code components via socket connection, that are necessary to get the client started. Hacked clients failing to complete these transaction will be detected and discarded.
- **Secure token verification**: an extra token can be exchanged between client and server (either SmartFox or the HTTP server) to add an extra layer of protection against forged clients.
- **Server-side signature:** similar to the "token technique" the server could sign each client binary with a special, time-limited signature that will expire within minutes. This way modified clients will fail to access the server because unable to provide a valid signature.

» Mobile Clients

Mobile clients should be regarded as quite secure because most phone and tables OSes such as iOS and Android provide a secure source of application distribution via their online stores. Additionally mobile apps are usually compiled to native code and signed by the author, offering an extra layer of security.

While this is valid for the official channels of distribution, it is still possible for malicious clients to use hacked phones or custom applications to attack the server.

Token signing is still possible, although less effective, while server-side signing is unfortunately not feasible. Another strategem is to require a valid phone number where a security token can be sent via text message and used by the client application as an additional identity check.

» Thick Clients

We put the classic thick client (e.g. downloadable .exe installer, usually larg size) in the last postion because it is probably the most vulnerable form of client.

Since all the executable code and dependencies are installed on the local machine any motivated hacker will be able to dissect the code, capture the network traffic and, with some patience, figure out possible ways for an attack.

The suggestions we have provided in this section also apply to this class of clients: code obfuscation, version control and validation tokens can all improve the overall security of the application and keep the common attacks at bay.

Conclusions

As with all things there is no perfect solution or golden rule to secure an online multiplayer application or game: each case will have it's own strength and weaknesses. The purpose of this paper is to dissect all the common ways in which an attacker can try to abuse of the system, from the lowest network stage up to the API level.

Being aware of the possible routes for an attack and the best countermeasures to use is the first step to making our applications more secure and less prone to misuse.

Finally we highly recommend to consult the official SmartFoxServer 2X documentation website for more details on the topics we have touched in this paper.

For further inquiries you can get in touch with us at: info@smartfoxserver.com